Universitat de Barcelona

Facultat de Matemàtiques i Informàtica

Departament de Matemàtiques i Informàtica

Grado en Ingeniería Informática

¿Lo perdiste? Una app colaborativa para Android que te ayuda a encontrar tus objetos perdidos.

Sergi Roge Sal

Trabajo Final de Grado

Tutor : Sergio Sayago Barrantes

Curso académico 2016-2017

Abstract

Finding a lost object can sometimes be a very tedious task. Moreover, no matter how hard you try to find that object, you might not be able to find it. Consequently, people tend to give up looking for lost objects. To make matter worse, when someone finds a lost object and has the intention to give it back to its owner, s/he does not know how to do it.

This TFG does not aim to provide 'the' solution to this problem, as doing so rests upon a lot of research. Instead, this TFG aims to help to keep the owner of a lost object in touch with the person who found it through an app wherein found or lost objects registered, by introducing their physical features, when they were found and where.
In order to find the lost object, it is mandatory that both the owner of the lost object and the person who found it register the object in the same way.

If there is a match, the application will notify both users. Upon a confirmation, both the owner of the object and the person who found it agree on how, when and where they can meet. This 'negotiation' is not carried out through the app.

In order to carry out this project and develop the app, we reviewed previous works and conducted semi-structured interviews with end-users to gather requirements. Once all the functionalities where narrowed, having in mind the regulation of the Civil Code that regulates lost objects in Spain, we developed the database, client and server side of the app.

The client side was developed as an application for Android devices, developing the main functionalities that allow the users to register lost and found objects.

The server side was implemented with a web server apache that contains the matching algorithm that matches lost and found items, and the database that contains all the information related to the users and objects.

Tabla de contenido

1. Introducción

Todo proyecto, en este caso Trabajo de Final de Grado (TFG), debe de responder a una pregunta importante: ¿Por qué has hecho este TFG ?
Como en todos los casos, obviamos la primera respuesta que es la obligatoriedad para obtener el título, Graduado en Ingeniería Informática. Este proyecto tiene, además, otra motivación, explicada en la siguiente sección.

1.1. Motivación

Estando una vez en una parada de autobús en Barcelona, me encontré unas gafas en el asiento de la parada. Desde luego esas gafas graduadas no me pertenecían intuyendo que esas gafas muy probablemente nunca regresarían a su propietario. Lo único que podría ayudar sería dárselas al conductor del autobús, para que éste las llevara a objetos perdidos de TMB.

Una vez en el autobús, y pensando en lo sucedido, mientras me dirigía a casa, realicé una búsqueda de aplicaciones que ayudaran a las personas a encontrar algún objeto que hubieran perdido utilizando mi smartphone. Las aplicaciones que encontré requieren acoplar un dispositivo físico al objeto susceptible de ser perdido, cuya función es indicar la ubicación geográfica del objeto.

Pensé que la utilidad de estas aplicaciones no tiene mucho sentido ya que se limita a todos tus objetos a los cuales le hayas adjuntado el dispositivo localizador. Siendo sinceros, no creo que vayamos a adjuntar a todas tus pertenencias dicho localizador.

Hablando con amigos y familia sobre el tema, sus comentarios me hicieron ver que a todos les ha pasado en algún momento de sus vidas que han perdido algo que no han recuperado. Mi experiencia en la parada del autobús y las opiniones de los míos, me motivaron a realizar este TFG.

1.2. Objetivos

El objetivo principal de este TFG es diseñar e implementar una aplicación móvil para ayudar a que las personas que hayan perdido un objeto personal lo puedan recuperar fácilmente.
Alcanzar dichos objetivos requiere, además cumplir los siguientes sub-objetivos:

1. Entender de qué manera, el proceso de búsqueda de objetos perdidos se puede mejorar mediante una app.
2. Diseñar una arquitectura que permita dar respuesta a las necesidades e intereses de los usuarios, además de permitir posibles futuras mejora.
3. Diseñar una base de datos para guardar la información de los usuarios y objetos.
4. Implementar una aplicación sencilla y usable para que los usuarios puedan registrar objetos perdidos y encontrados.
5. Implementar un servidor acorde a las necesidades de la aplicación.

1.3. Problema, relevancia y trabajos relacionados

Este TFG aborda un problema real de considerable importancia. ¿Quién no ha perdido alguna vez un objeto, ya sean las llaves de casa, del coche o unas gafas de sol? Y, una vez que nos hemos dado cuenta de que hemos perdido dicho objeto, ¿Cómo lo podemos recuperar – si es que finalmente lo hacemos? Por otro lado, también es razonable pensar que en más de una ocasión nos hemos encontrado objetos por la calle, y hemos pensado cómo devolvérselos a sus dueños, sin saber muy bien cómo.

1.4. Solución propuesta

En este TFG, proponemos una aplicación colaborativa para smartphone que sirva de nexo entre la persona que ha perdido un objeto personal y la persona que lo ha encontrado, y tiene la intención, pero no sabe cómo, devolver dicho objeto a su dueño.

En la aplicación se registran objetos perdidos y encontrados, y si las características de los objetos son muy similares, así como la fecha y la zona donde se registraron tanto la pérdida como la recuperación, la aplicación pone en contacto a las dos personas para que la devolución del objeto perdido se pueda efectuar.

La inmediatez de utilizar el móvil al darse cuenta de que has perdido algo es el motivo principal por el cual hemos optado por diseñar una aplicación para móvil y no, por ejemplo, una página web, aunque no es una opción a descartar en un futuro.

Esta aplicación se distingue de las ya existentes en el punto en que no fuerza al usuario a adjuntar un localizador a cada uno de sus objetos personales susceptibles de ser perdidos, ya que nuestra idea de aplicación se basa en la colaboración ciudadana, sin embargo, ambas intentan solucionar el problema de encontrar un objeto perdido.

1.5. Resumen del trabajo desarrollado y principales resultados

Hemos realizado un estudio del estado del arte, para obtener una visión de la existencia, o no, de aplicaciones similares. Hemos realizado entrevistas a usuarios potenciales para entender qué hacen cuando pierden n objeto, o se encuentran con uno, y conocer su actitud hacia una aplicación para móvil para gestionar este aspecto. Hemos hablado con expertos en el ámbito de la legalidad. Estas conversaciones indicaron las obligaciones de aquellos que encuentran un objeto perdido, y se han incorporado en el diseño de la aplicación. Hemos utilizado herramientas, para diseñar un servidor con la base de datos acorde a las necesidades de la aplicación y ésta se ha diseñado acorde a las necesidades de los usuarios y nuestras propias ideas.
El resultado final este TFG es una app, que ha seguido una serie de etapas de desarrollo del software. Hemos diseñado una aplicación teniendo en cuenta la previa recogida de requerimientos, que se realizó a usuarios potenciales con y sin conocimientos sobre la informática. Hemos analizado tecnologías existentes para elegir aquellas que nos ayudasen a implementar correctamente todas las funcionalidades deducidas en base a los requerimientos. Hemos implementado la parte del cliente y la parte del servidor junto a la base de datos y documentado en una memoria (el presente documento) todo el trabajo realizado durante el proyecto.

2. Diseño de la aplicación

Antes de diseñar una aplicación (y, de hecho, cualquier tecnología interactiva / digital dirigida a personas), es una buena práctica adoptar un punto de vista centrado en el usuario. En este TFG, se decidió que antes de diseñar la aplicación – que era una decisión
que ya tomamos, y que hemos justificado anteriormente – debíamos
entender la perspectiva del usuario final. Para ello, realizamos una entrevista semi-estructurada a un
conjunto de usuarios, que detallamos en el apartado 2.1.

Los aspectos (éticos y) legales suelen ser importantes en el desarrollo 'profesional' de una aplicación. En este caso, nos encontramos con un aspecto legal de especial relevancia: el asunto de los objetivos perdidos y encontrados está regulado por el código civil. En el apartado 2.2 presentamos cómo abordamos este aspecto.

En el apartado 2.3 mostramos el diseño en bajo nivel de las principales
funcionalidades de aplicación. El diseño en alto nivel se encuentra en la sección X de la memoria del TFG.

2.1. Entrevista semi-estructurada: usuarios, preguntas y resultados

Teníamos ideas de cómo debía ser la aplicación, pero era necesario realizar una entrevista personal a potenciales usuarios, para saber su opinión y recoger sus propias ideas. Los usuarios fueron conocidos y familiares, personas de fácil acceso, algunos con conocimientos relacionados con la informática y otros no. De esta manera hemos intentado abarcar a un grupo heterogéneo de usuarios.
Consideramos que para diseñar nuestra aplicación, era necesario saber cómo sus usuarios gestionan la pérdida de un objeto personal, acciones que han tomado, etc. Para ello decidimos realizar una entrevista semi-estructurada.
La entrevista se ha dividido en bloques. Las preguntas que se muestran a continuación son de carácter personal y abordan el problema del TFG.

¿Has perdido alguna vez algún objeto?
¿Qué objeto?
¿Lo has encontrado?
¿Has hecho algo para encontrarlo?
¿Te has encontrado alguna vez algún objeto perdido por la calle ?
¿Qué has hecho con ese objeto ?

¿Si pudieras perder 5 minutos de tu tiempo para facilitar que la persona que ha perdido el objeto, lo pudiera recuperar, lo harías?
¿Cómo lo harías?

Los resultados obtenidos, indicaron que la pérdida de un objeto personal es habitual entre los entrevistados, aunque no es algo que suela suceder con mucha regularidad.

Las preguntas que siguen a continuación están enfocadas ya en la idea de aplicación y pretenden identificar si los entrevistados conocían alguna aplicación existente para solucionar el problema.

¿Conoces alguna aplicación para encontrar objetos perdidos ?
¿Si hubiera una aplicación para móvil que te ayudara a encontrar tu objeto perdido, la utilizarías?

Nuestros usuarios no conocían la existencia de ninguna aplicación que les ayudara a encontrar objetos perdidos. este resultado sugiere que nuestra app puede ser de gran utilidad. Para profundizar en este aspecto, les mostramos algunos prototipos de muy bajo nivel (dibujos en papeles) para guiar la entrevista a aspectos más técnicos y de diseño.

¿Qué te parecería una aplicación similar a esta?
¿La usarías?
¿Le añadirías o le quitarías algo?

Todos los usuarios se mostraron curiosos e interesados en la aplicación, proponiendo algún cambio en ella. Todos coincidieron en que utilizarían la aplicación en caso que perdieran o encontraran algún objeto.

Los resultados de la entrevista a los usuarios se pueden encontrar en el anexo apartado 8.1.

2.2. Problema legal y solución

Los aspectos éticos y legales son importantes en el desarrollo de una app. En este caso, nos encontramos con un aspecto legal de especial relevancia, el código civil regula los hallazgos.

Una vez realizada la entrevista, y en las primeras fases del diseño, un familiar, del autor del TFG, nos comentó que existía una regulación en el código civil español (Artículo 615), con ciertas especificaciones por el código civil catalán (Artículo 542-22. Hallazgos.) relacionada con los objetos perdidos.

Artículo 615 Código Civil Español

El que encontrare una cosa mueble, que no sea tesoro, debe restituirla a su anterior poseedor. Si éste no fuere conocido, deberá consignarla inmediatamente en poder del Alcalde del pueblo donde se hubiese verificado el hallazgo. (...)

Artículo 542-22. Hallazgos Código Civil de Cataluña.
(...)
2. Si los propietarios son desconocidos, el hallazgo debe notificarse al ayuntamiento del lugar donde se ha hecho, el cual debe hacerlo público por medio de un edicto, debe depositar la cosa durante el plazo de seis meses en el establecimiento que determine y debe notificarlo a las entidades públicas pertinentes si las características del hallazgo lo requieren.
(...)

Los artículos entraban en conflicto de la aplicación, ya que inicialmente ésta parece incentivar a quedarte el objeto perdido. Sin embargo hablando con expertos en aspectos legales, nos indicaron que, un individuo que encontrase un objeto perdido, no está obligado como primera decisión y acción a llevar el objeto al ayuntamiento - siempre y cuando, exista intención de devolver dicho objeto. Por tanto, pudimos seguir adelante con la app.

Como se verá más adelante, la app muestra información relacionada con los artículos del código civil y se indica que la aplicación es una opción alternativa a llevar el objeto perdido, al ayuntamiento del municipio donde se encontró dicho objeto.

2.3. Diseño de las principales funcionalidades

Teniendo en cuenta lo aprendido al recoger y analizar los requerimientos y la idea de la aplicación, nos dimos cuenta de las funcionalidades que debía tener, continuación las detallamos.

2.3.1. Registrarse como usuario

El registro de los usuarios es una funcionalidad importante para gestionar “quién ha perdido qué” y “quién ha encontrado qué”. Hay aplicaciones que te permiten el acceso mediante un pseudónimo, pero sin que haya un previo registro, pensamos que eso no era la solución que buscábamos, queríamos algo más sólido, por lo que nos decantamos por la solución de registro y acceso de usuarios. Algunos entrevistados propusieron mas campos en el registro de usuarios, pero estas propuestas no afectaban a la principal funcionalidad de la aplicación, pero podrían ser utilizadas en un futuro, en el apartado 6.1 Mejoras a realizar en siguientes versiones lo explicamos con más detalle.

Para registrarse será necesario un correo electrónico, la contraseña y un nombre de usuario. Existe un control mediante el uso de expresiones regulares para el registro de los usuarios. Se permiten caracteres a-A, z-Z y números del 0 al 9 en todos los campos, además en el correo se comprobará si sigue una estructura corriente de correo electrónico (correo@electronico.x) donde x tendrá longitud máxima de 3 caracteres. A continuación se muestra el mockup de la pantalla de registro de usuarios.

Figura 1. Mockup de la pantalla de registro de usuarios

2.3.2. Identificarse como usuario

Una vez un usuario se ha registrado, para comenzar a utilizar la aplicación deberá identificarse. Deberá introducir el correo electrónico y la contraseña con los cuales se registró.

Al identificarse correctamente, se redirige a la pantalla principal de la aplicación.
Como hemos explicado en la pantalla de registro de usuarios, existe también para la pantalla de acceso a la aplicación un control utilizando expresiones regulares también, permitiendo los mismos caracteres.

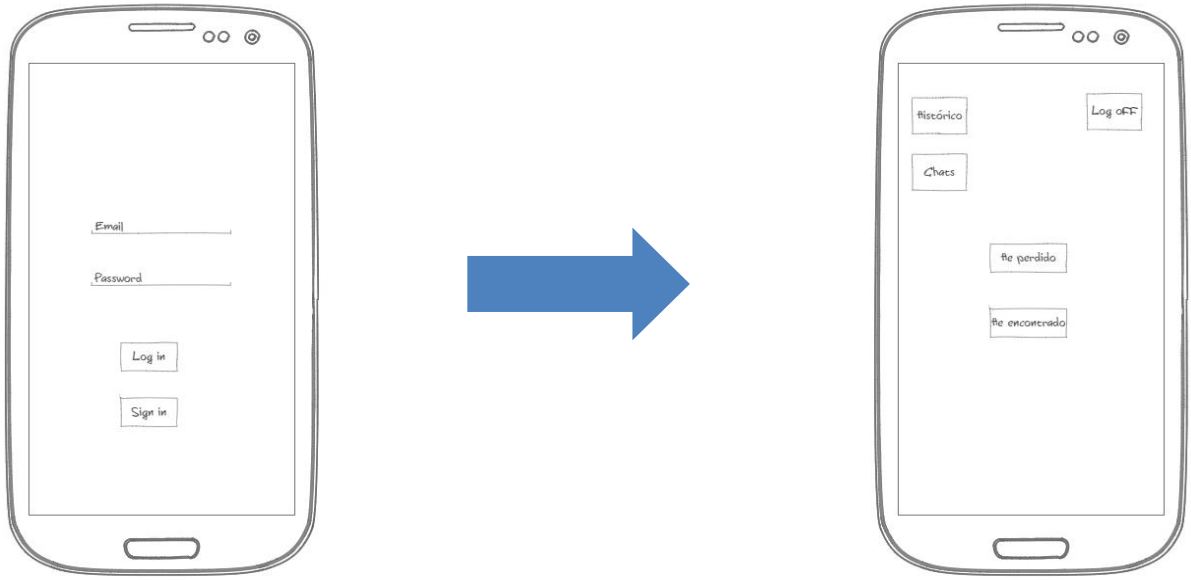

Figura 2. Mockup de las pantallas de acceso a la aplicación.

2.3.4. Registrar un objeto perdido / encontrado

La funcionalidad de registro de objetos es la principal de las funcionalidades de nuestra aplicación, al presentar nuestra idea a los entrevistados, todos coincidieron que encontraban interesantes los pasos a seguir, primero introduciendo características físicas y luego las ubicaciones. Nos indicaron también la opción de permitir añadir más de un material y color al objeto registrado, pero consideramos que para que fuera más sencilla la implementación del cliente y el algoritmo de coincidencia de objetos, registraríamos esa petición como una mejora de las siguiente versiones.
Para registrar un objeto perdido / encontrado, se pulsa el botón correspondiente, para registrar un objeto perdido, el botón 'He perdido', para un objeto que se ha encontrado, 'He encontrado'. Una vez pulsado, la siguiente pantalla (pantalla 1) nos permitirá introducir las características físicas del objeto. Por ejemplo, el tipo de objeto, el color, la marca y el material, además, también permite indicar cuándo se perdió el objeto.
Una vez introducidos estos datos y pulsando el botón 'Siguiente', la siguiente pantalla permite introducir el lugar en el que hemos perdido / encontrado el objeto (pantalla 2). Dicha pantalla nos permite añadir hasta 3 ubicaciones de Google Maps.
Una vez introducido el lugar, si el objeto es un objeto encontrado, pulsaremos el botón de 'Guardar' y la aplicación redirigirá al usuario a la pantalla principal.

Si por el contrario el objeto registrado es un objeto perdido, el botón a pulsar es ‘Siguiente’, y la aplicación nos redirige a la pantalla que nos permite añadir información adicional (pantalla 3), cualquier descripción del objeto que permita a quien lo ha encontrado determinar si es o no el objeto.(Esta información no se utiliza en el algoritmo de matching, simplemente es de soporte)

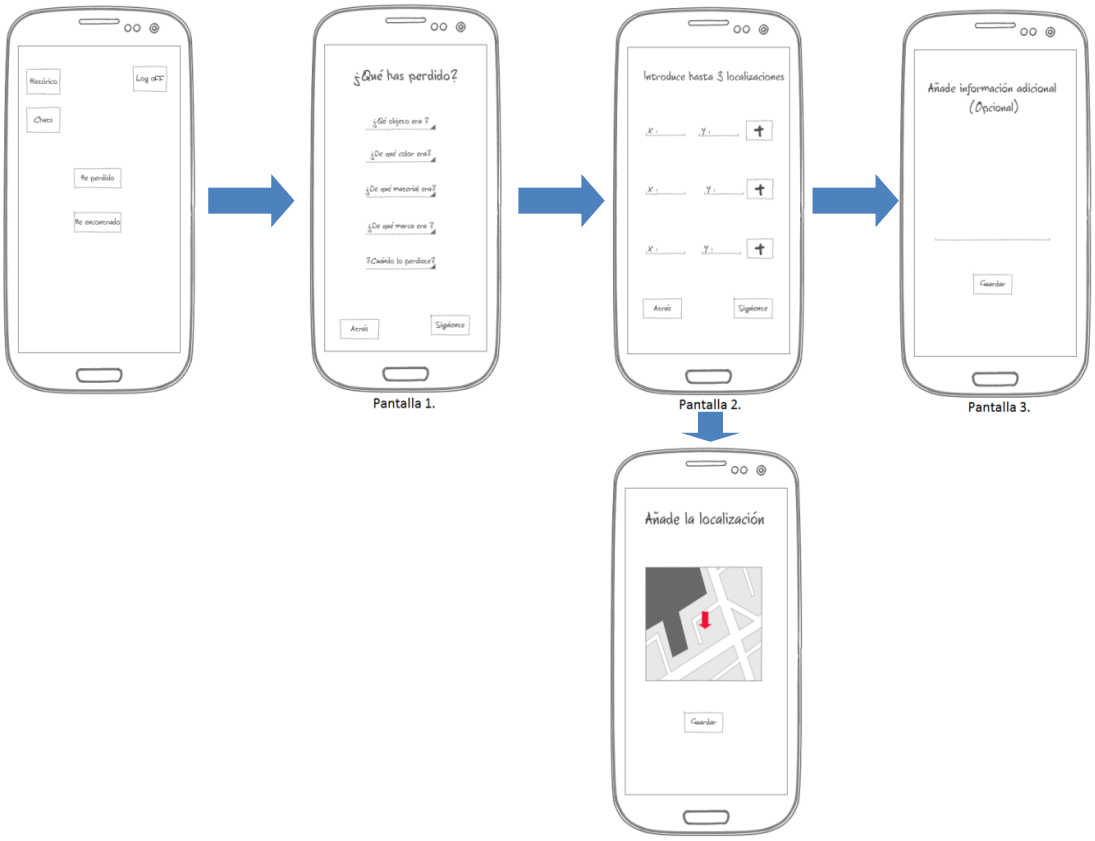

Figura 3. Mockup de la funcionalidad de registro de objetos.

2.3.5. Consultar un objeto registrado

La aplicación le permite consultar todos los objetos registrados anteriormente a un usuario, pulsando el botón ‘Archivo’ en la pantalla principal se redirige al usuario a la pantalla de archivo.
Estando en dicha pantalla, pulsando cualquier elemento de la lista, se le permite al usuario ver de forma más detallada el objeto seleccionado.

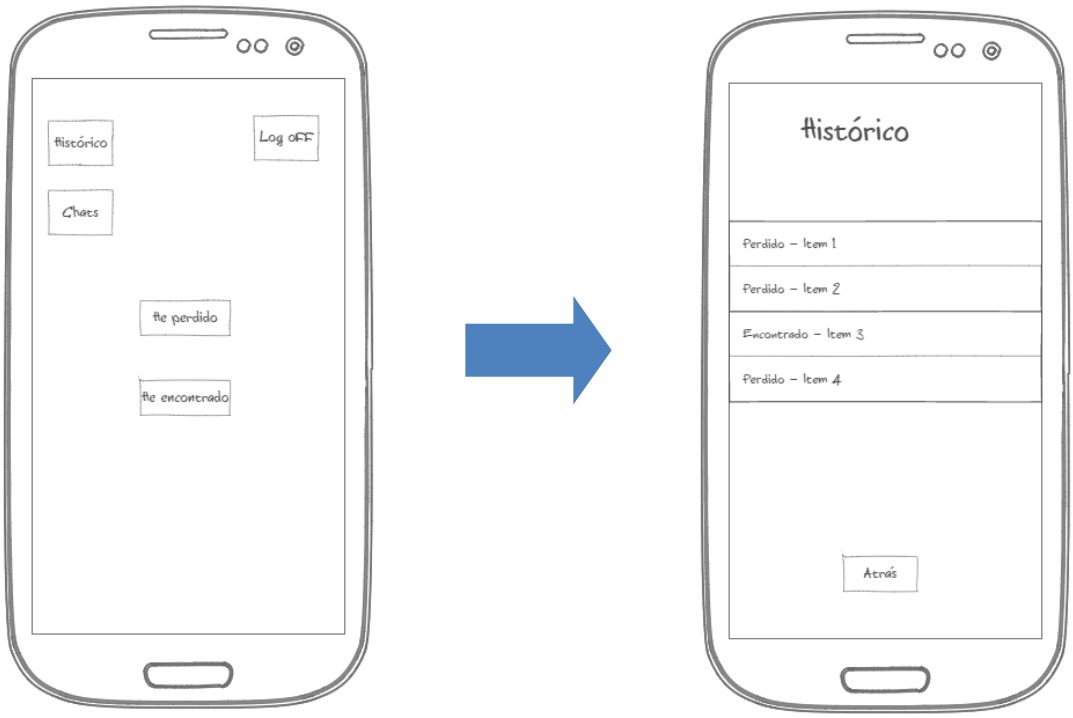

Figura 4. Mockup de la funcionalidad de consulta de objetos registrados.

2.3.6. Confirmar objeto encontrado

Una vez el sistema determina que un objeto encontrado por un usuario puede ser similar a un objeto perdido por otro, la aplicación notifica al usuario que ha encontrado el objeto, indicándole que, posiblemente, posee el objeto de alguien.
Se le envía una notificación, mostrando la información adicional que el usuario que registró el objeto perdido añadió (Pantalla 3 del apartado 2.3.4 Registrar un objeto perdido / encontrado.).
El usuario decidirá, pulsando el botón de confirmación o el de cancelación, si la información mostrada coincide con el objeto que encontró o no.

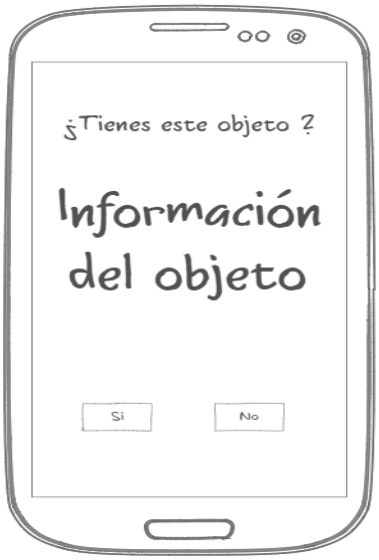

Figura 5. Mockup de la funcionalidad de confirmación de objeto.

2.3.7. Chat entre usuarios

Una vez el usuario ha confirmado que posee el objeto perdido de otro usuario, el sistema inicializará un chat entre ambos usuarios.
Para consultar los chats, el usuario debe pulsar el botón de chats en la pantalla principal, se le redirigirá a la pantalla donde están todos los chats (Pantalla 1), pulsando cualquiera, se abrirá el chat con dicho usuario (Pantalla 2).

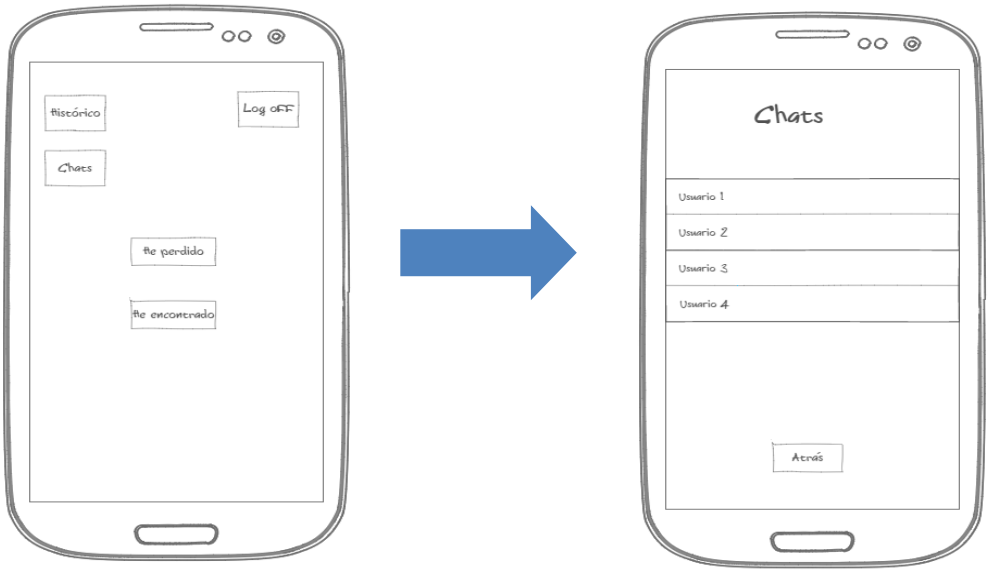

Figura 6. Mockup de la funcionalidad de chat entre usuarios.

3. Desarrollo de la aplicación

La sección 3.1 resume los principales rasgos de la metodología de desarrollo utilizada. La sección 3.2 presenta un estudio de tecnologías existentes y relacionadas con el objetivo del TFG. Posteriormente, detallamos el desarrollo del cliente (3.3), del servidor (3.4) y de la base de datos(3.5)

3.1. Metodología inspirada en SCRUM

En cuanto a la metodología de desarrollo, nos hemos inspirado en metodologías ágiles, ya que aporta calidad al software.
Inicialmente se diseñó una tabla de Scrum, dividiendo el trabajo venidero en tareas. Sin embargo, consideramos que el TFG no seguiría un Scrum al 100%, pues no sabíamos el tiempo que llevaría implementar la arquitectura de la aplicación y del servidor.
Scrum es muy útil haciendo trabajo en equipo, dividiendo tareas en diferentes miembros, de manera individual es una simple organización del trabajo.
En consecuencia, dividimos las tareas en funcionalidades, sin utilizar sprints.
También, en entornos profesionales y de de trabajo en grupo, los sprints de Scrum suelen finalizar con una muestra al cliente o al product owner del estado del software de las tareas que se han hecho vs las tareas planificadas, pero en este proyecto, era difícil plantear unas tareas muy concretas durante un sprint y, al final de éste, mostrarlas al cliente final.

3.2. Tecnologías y herramientas utilizadas

La aplicación se puede dividir en dos partes la aplicación, la parte del cliente y la parte del servidor, para cada una de ellas se han utilizado tecnologías diferentes.

Para la parte del cliente, ya como hemos comentado en la solución propuesta, se trata de una aplicación para móvil. Nos hemos decantado por Android, ya que es lenguaje Java, es conocido y hay mucha documentación. Utilizamos la versión 6.0 Marshmallow, API 23 ya que uno de nosotros (el autor) realizó un curso de Android en esa versión, así que podríamos profundizar en aspectos de aprendizaje.

Para la parte del servidor, había varias propuestas, unas desconocidas y otras más conocidas para nosotros. Teníamos conocimiento de la existencia de MongoDB, un servidor Python que, junto con el uso postgreSQL para diseñar la base de datos podría ser una potente herramienta, pero por otro lado, ya conocíamos XAMPP, un servidor

web apache, utilizando PhP como lenguaje para el desarrollo del servidor y utilizando PhPMyAdmin para el diseño de la base de datos, en MySQL.

Finalmente nos decantamos por los sistemas conocidos, ya que debido a la limitación del tiempo, nos interesaba ser capaces de desarrollar la app en un entorno tecnológico más familiar, y a su vez profundizar en nuestro conocimiento de estas tecnologías.

Para preparar el servidor, primeramente tuvimos que descargarlo de la página oficial (https://www.apachefriends.org/es/index.html) siguiendo los pasos de la instalación conseguimos instalarlo correctamente. Una vez instalado, ya pudimos empezar a utilizar las herramientas que el instalador contiene, PhPMyAdmin como un portal para la gestión de la base de datos y también empezar a desarrollar el código del servidor, utilizando PhP como lenguaje de programación. Nos hemos decidido por el uso de PhP para la implementación del servidor ya que es un lenguaje sencillo y nos debería permitir la posible ampliación del sistema hacia un cliente web además de la aplicación para móvil.

Además de estas herramientas de implementación, hemos utilizado GitHub para el versionado del código, siendo https://github.com/SergiRoge/TFG el enlace al código del cliente y https://github.com/SergiRoge/ServerSide el enlace al código del servidor, donde se puede ver todas las subidas de código que se han ido realizando durante la implementación.

3.3. Implementación del cliente

3.3.1. Paquetes Java

Como hemos comentado, la implementación del cliente se ha hecho usando la API 23 de Android. En el diseño de los mockups ya se muestran las pantallas, así que en este punto explicaremos su funcionamiento por debajo.
Hemos organizado las clases en los siguientes paquetes, ya que es una buena práctica en el desarrollo del software tener el código organizado y estructurado.
Además, en cada apartado hemos comentado algunas partes del código que nos han parecido interesantes.

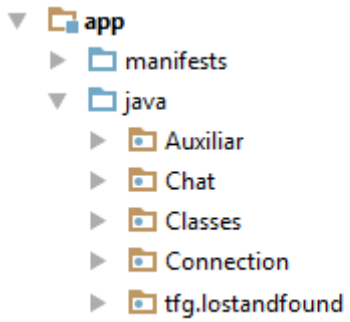

Figura 7. Estructura de los paquetes del cliente.

A continuación, mostramos el diagrama de clases de las principales clases de la aplicación: usuarios, objetos y coordenadas.
Un usuario contiene varios objetos, y éstos contienen hasta tres coordenadas, las tres clases heredan de SQLObject, ésta a su vez de Thread, para poder ejecutar la tarea en paralelo y Serializable para poder añadir las clases como extras a través de intents, para poder ejecutar los métodos que hemos implementado en la clase Connection.

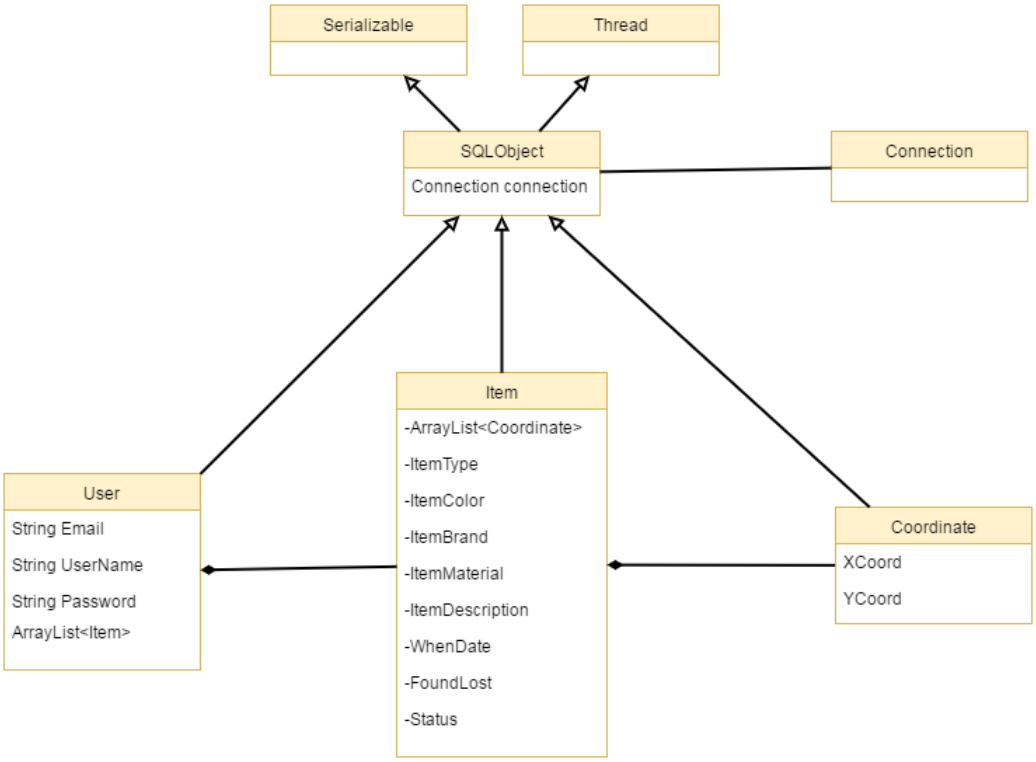

Figura 8. Diagrama de las principales clases de la aplicación

3.3.1.1. Paquete Auxiliar

El paquete Auxiliar contiene aquellas clases que sirven de soporte para otras, así como clases y métodos estáticos para poder ser llamados desde cualquier otra clase del código.

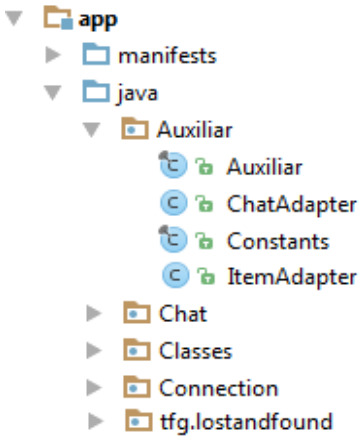

Figura 9. Estructura del paquete Auxiliar.

- *Auxiliar.java*, contiene métodos estáticos para ser utilizados en todo el contexto de la aplicación.
  En esta clase podemos destacar el siguiente fragmento de código, que contiene el control de expresiones regulares que se utiliza en los campos de usuario, correo y contraseña que se encuentran en las pantallas de registro de usuarios y acceso a la aplicación.

```
public static boolean verifyNormalField(String pstrValue)
{
    //Un pequeño control de longitud, si es menor de 4 caracteres
    //devolvemos falso, el campo no es válido
    if(pstrValue.trim().length() < 4)
    {
        return false;
    }
    //Creamos el patrón que contiene las letras del abecedario
    //mayúsculas y minúsculas y los numeros del 0 al 9.

    Pattern p = Pattern.compile("^[a-zA-Z0-9_-]$");

    //Devolvemos el string pasado por parametro
    //contiene únicamente los carácteres que permite
    //el patrón creado más arriba, true.
    //Devolvemos falso en cualquier otro caso
    Matcher m = p.matcher(pstrValue);
    return m.matches();
}
```

Figura 10. Fragmento de código que verifica si los campos informados contienen caracteres no permitidos.

- *ChatAdapter.java*, clase auxiliar para ayudar a la visualización de la lista de chats en la pantalla ChatsActivity.java.

- *Constants.java*, clase que contiene las constantes utilizadas en la aplicación.

- *ItemAdapter.java,* clase auxiliar para ayudar a la visualización de la lista de items perdidos y encontrados en la pantalla Archive.java.

3.3.1.2. Paquete Chat

El paquete chat contiene todas las clases que intervienen en la implementación del chat entre usuarios. Por el momento, solamente contiene la clase Chat.java. Aún así, hemos decidido crear el paquete para futuras ampliaciones.

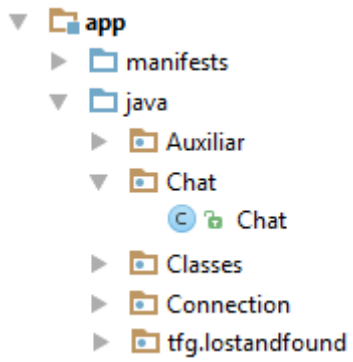

Figura 11. Estructura del paquete Chat.

- *Chat.java,* Contiene los atributos y métodos para poder guardar y cargar los chats de los usuarios.

3.3.1.3. Paquete Classes

El paquete Classes contiene los principales elementos de la aplicación, los usuarios y los objetos con sus coordenadas, entre otras clases.

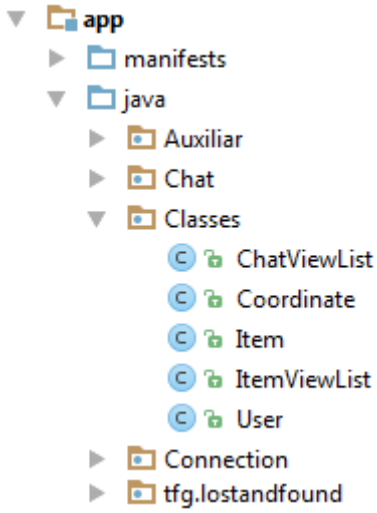

Figura 12. Estructura del paquete Classes.

- *ChatViewList.java,* contiene toda la información que se mostrará en cada uno de los elementos que existan dentro del viewlist de la pantalla de chats.

- *ItemViewList.java,* el mismo concepto que la clase ChatViewList.java, pero en este caso contiene la información para el viewlist de la pantalla de archivo, donde se muestran los objetos registrados de un usuario.

- *Coordinate.java,* la clase que contiene los atributos y métodos de cada coordenada.

- *Item.java,* la clase que contiene los atributos y métodos de cada item.
  Destaco en esta clase el método que guarda un objeto en el servidor.
  El método prepara todos los parámetros que el webservice necesitará, en este caso todas las características físicas del objeto, el usuario que lo registra y las ubicaciones. Después se llama al método pasando por parámetros la URL donde está el webservice y las características del objeto.

```
public int save(User user) throws IOException, InterruptedException {

    //Preparamos todos los parámetros que pasaremos por la URL
    String content = "";
    content += "type="+ strItemType + "&" +
               "color="+ strItemColor + "&" +
               "brand="+ strItemBrand + "&" +
               "material="+ strItemMaterial + "&" +
               "when="+ intWhen+ "&" +
               "description="+strDescription + "&" +
               "status="+intStatus + "&" +
               "foundLost="+strFoundLost;

    Coordinate coord = new Coordinate();

    //Preparamos las coordenadas del objeto
    //para pasarlas por parámetro también
    for(int i = 0; i < arrayListCoordsAdded.size(); i++)
    {
        coord = arrayListCoordsAdded.get(i);
        content +=  "&coordX"+i+"=" + coord.getDblXCoordinate() + "&coordY"+i+"="
                + coord.getDblYCoordinate();
    }
    //Añadimos el correo del usuario para
    //pasarlo por parámetro
    content += "&user="+user.getStrEmail();

    //Y finalmente el número de cooordenadas que contiene el objeto
    content += "&coordsnumber="+arrayListCoordsAdded.size();

    //Llamamamos al método que llamara al webservice y guardará el objeto
    //según los parámetros enviados
    String strReturn =  ExecuteQuery(URL_SAVE_ITEM, content);

    return 1;
}
```

Figura 13. Fragmento de código del método save() de la clase Item.java.

- *User.java,* la clase que contiene los atributos y métodos de cada usuario.
  Como método interesante, hemos considerado mostrar el método que recupera la información del servidor, dado un correo y una contraseña, es decir, al acceder a la aplicación.

```
public int retrieveUserData() throws IOException, InterruptedException, JSONException
{
    //Preperamos los parámetros necesarios
    String content = "";
    content +="password=" + strPassword + "&" +
            "email=" + strEmail;

    //Llamamos al webservice y recibimos toda la información
    //de el usuario en un string en formato JSON.
    String JSONstrReturn =  ExecuteQuery(URL_CHECK_USER, content);

    //LLamamos al método que obtendrá la información devuelta
    //por el servidor del usuario
    return fillUserFields(JSONstrReturn);


}
```

Figura 14. Fragmento de código del método retrieveUserData() de la clase User.java.

3.3.1.4. Paquete Connection

El paquete Connection contiene las clases necesarias para establecer conexión con el servidor y comunicarse cuando se guardan o cargan datos de la base de datos.

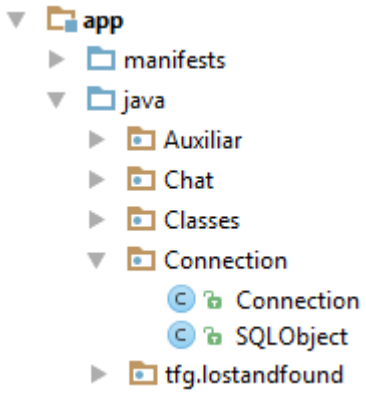

Figura 15. Estructura del paquete Connection..

- *SQLObject.java,* clase intermedia que sirve de capa entre las clases principales, Item, Coordinate, User y Chat, y la clase Connection, su finalidad es encapsular los métodos para facilitar la programación.
  El método ExecuteQuery hemos considerado interesante de mostrar y comentar.

```
public String ExecuteQuery(String URL, String contents) throws IOException, InterruptedException
{
    //Creamos una nueva conexión con el servidor
    //y le pasamos la dirección del webservice y
    //los parámetros de éste
    connection = new Connection(URL, contents);

    //Ejecutamos el hilo que se encargará
    //de llamar al servidor
    connection.start();

    //Esperamos a que este muera
    connection.join();

    //Una vez el hilo acabó su ejecución
    //devolvemos lo que el webservice nos ha
    //devuelto
    return connection.SQLResult;
}
```

Figura 16. Fragmento de código del método ExecuteQuery() de la clase SQLObject.java.

- *Connection.java,* clase que realizará la conexión con el servidor, se encargará de realizar la petición al servidor mediante el uso del método POST y llamará al webservice correspondiente con los atributos necesarios.
  A continuación mostramos cómo se hace la conexión con el servidor.

Primeramente, el constructor de la clase Connection, asignando valores a la URL y a el contenido, que serán los parámetros, y después creando la conexión con el servidor.

```
public Connection(String pstrURL, String pstrContents) throws IOException
{

    //Inicializamos la URL con la que nos viene por parámetro
    this.strURL = new URL(pstrURL);

    //El String que contiene toda la información necesaria
    contents = pstrContents;

    //Creamos la conexión con la URL
    httpsURLConnection = (HttpURLConnection)strURL.openConnection();
}
```

Figura 17. Fragmento de código del constructor de la clase Connection.java.

El siguiente método, execute(), es el método que se llama cuando ejecutamos el thread de conexión, mostrado en la clase *Connection.java* de este mismo apartado.

```
public void execute() throws IOException
{
    //Inicializamos parámetros de conexión y método POST
    httpsURLConnection.setReadTimeout(10000);
    httpsURLConnection.setConnectTimeout(15000);
    httpsURLConnection.setRequestMethod("POST");
    httpsURLConnection.setDoInput(true);
    httpsURLConnection.setDoOutput(true);
    httpsURLConnection.setRequestProperty("Content-Length", Integer.toString(contents.length()));

    //Realizamos la conexión
    httpsURLConnection.connect();

    //Creamos el buffer de escritura de salida, en dirección al servidor
    OS = httpsURLConnection.getOutputStream();
    bufferedWriter = new BufferedWriter(new OutputStreamWriter(OS, "UTF-8"));

    //Escribimos en el buffer los parámetros, vaciamos y cerramos la conexión
    bufferedWriter.write(contents);
    bufferedWriter.flush();
    bufferedWriter.close();
    OS.close();

    //Creamos el buffer de escritura de entrada, lo que nos devuelve el servidor
    IS = httpsURLConnection.getInputStream();

    //Guardamos en SQLResult el resultado de lo que nos ha devuelto el servidor y cerramos
    //el buffer
    SQLResult = convertinputStreamToString(IS);
    IS.close();

    //Cortamos la conexión con el servidor
    httpsURLConnection.disconnect();

}
```

Figura 18. Fragmento de código del método execute() de la clase Connection.java.

3.3.1.5. Paquete Main

Paquete principal del proyecto, se encuentran todas las clases java que se relacionan con cada activity o widget existente en el proyecto. Se encuentran también los servicios que se lanzarán durante la ejecución de la aplicación, inicialmente para éstos se creó un paquete de servicios, por si en un futuro fuera necesario crear más, pero por problemas en el fichero manifest, finalmente se colocaron en este paquete.

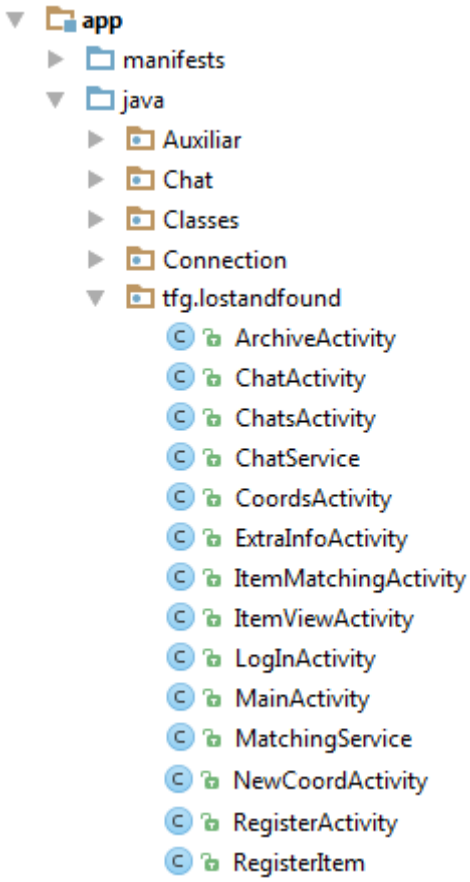

Figura 19. Estructura del paquete principal.

- *LogInActivity.java,* clase java que relacionada con el fichero activity_log_in.xml, permite a un usuario acceder a la aplicación con un correo y una contraseña, también permite el registro de usuarios pulsando al botón de registro de usuarios.

- *RegisterActivity.java,* clase java que permite registrarse como usuario, relacionada con el fichero activity_register.xml, rellenando los campos pertinentes, se permitirá al usuario registrarse para poder acceder a la aplicación.

- *MainActivity.java,* clase java principal. Permite al usuario registrar un objeto

perdido o un objeto encontrado, además de consultar el archivo de objetos registrados, acceder a los chats iniciados con otros usuarios, y también desconectarse de la aplicación.
Esta clase contiene dos clases privadas, MatchingReceiver y ChatReceiver, que corresponden a dos clases de apoyo para los servicios de matching y de chat, ambas heredan de BroadcastReceiver, que explicaremos más adelante.

- *RegisterItem.java,* primera pantalla de la secuencia de registro de objetos, relacionada con el fichero activity_register_item.xml. Permite rellenar características físicas del objeto y avanzar a la siguiente pantalla.

- *CoordsActivity.java,* clase java que relaciona el fichero activity_coords.xml. Segunda pantalla del proceso de registro de objetos, permite añadir y quitar coordenadas a un objeto, redirigiendo a la clase NewCoordActivity.java.

  En el siguiente fragmento de código, mostramos la implementación del diferente comportamiento del botón de siguiente o guardar dependiendo de si el usuario registra un objeto perdido u objeto que ha encontrado respectivamente. Además consideramos interesante mostrar el mensaje que aparece al registrar el objeto.

```
//Si estamos registrando un objeto perdido, el botón tiene como texto 'Siguiente'
//y al pulsarlo nos redirige a la pantalla de información adicional.
if(item.getStrFoundLost().equals("Lost"))
{
    Intent I = new Intent(CoordsActivity.this, ExtraInfoActivity.class);
    I.putExtra("Item", item);
    startActivity(I);
}
//De la otra manera, registramos un objeto que hemos encontrado, el texto del botón
//indica 'Guardar' y el sistema nos redirigirá a la pantalla principal al pulsarlo
else if(item.getStrFoundLost().equals("Found"))
{
    //Pero antes, nos mostrará el mensaje :
    //
    //"Recuerda que siempre que puedes llevar el objeto que has encontrado al
    //ayuntamiento del municipio donde lo encontraste."
    //Tal y como hemos explicado, haciendo referencia a la regulación del código
    //civil.

    showMessageError(CoordsActivity.this,ITEM_REGISTERED_ALERT);

    Intent I = new Intent(CoordsActivity.this, MainActivity.class);
    I.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TOP);
    I.putExtra("FROM"," ");

    I.putExtra("Item", item);
    finishAffinity();
    startActivity(I);
}
```

Figura 20. Fragmento de código del comportamiento del botón siguiente/guardar de la pantalla de coordenadas.

- *NewCoordActivity.java,* clase java relacionada con el fichero activity_new_coord.xml, es una activity de google maps.
  De la siguiente manera, mostramos cómo hemos implementado la gestión de la inserción de ubicaciones de manera sencilla e intuitiva para los usuarios.
  Al mover la cámara se actualiza la posición del centro del mapa y se coloca el marcadorde dicha zona.

```
//Listener que se activa al mover la cámara, es decir
//al mover el mapa
mMap.setOnCameraChangeListener(new GoogleMap.OnCameraChangeListener() {
      public void onCameraChange(CameraPosition arg0) {

          //Se limpia el mapa
          mMap.clear();

          //Creamos un marcador y lo situamos, virtualmente en el centro
          final MarkerOptions marker = new MarkerOptions().position(arg0.target);

          //Nos quedamos con las coordenadas, redondeándolas a 5 dígitos.
          Double x = (double) Math.round(marker.getPosition().latitude * 10000d) / 10000d;
          Double y = (double) Math.round(marker.getPosition().longitude * 10000d) / 10000d;

          //Seteamos el valor de las coordenadas en ese punto en
          //ambos textview que hay en la pantalla
          txtXCoord.setText("" + x);
          txtYCoord.setText("" + y);

          //Añadimos el marcador en el mapa, en el centro
          mMap.addMarker(marker);
      }
  });
```

Figura 21. Fragmento de código del listener al mover la cámara en el mapa de google maps al añadir ubicaciones.

- *ExtraInfoActivity.java,* la clase cuyo fichero xml es activity_extra_info.xml, y se encarga de recoger la información adicional de la última pantalla de registro de un objeto perdido.

- *ArchiveActivity.java,* clase java relacionada con el fichero activity_archive.xml. Gestiona la pantalla de archivo, inicializando componentes y listeners de aquellos componentes que lo requieran. Se encarga de rellenar la lista de objetos que se mostraran en la pantalla.

- *ItemViewActivity.java,* clase java que muestra la información de un objeto cuando se pulsa en la lista de objetos registrados en la pantalla del archivo, relacionada con el fichero activity_item_view.xml.

- *ChatsActivity.java,* clase java relacionada con el fichero activity_chats.xml. Se encarga de la pantalla de chats, inicializando componentes y listeners de aquellos componentes que lo requieran y de inicializar la lista de chats.

- *ChatActivity.java,* clase java relacionada con el fichero activity_chat.xml.

- *MatchingService.java,* servicio encargado de consultar al servidor si existe algún objeto encontrado que coincida con un objeto perdido. Irá realizando consultas a las tablas pertinentes, una vez existe coincidencia, se lanza una notificación que avisa al usuario

- *ItemMatchingActivity.java,* clase java que gestiona la pantalla que se muestra al usuario que registró un objeto encontrado cuando hay coincidencia con otro perdido, mostrando además la información adicional que el usuario añadió al registrar dicho objeto perdido.
- El usuario tendrá dos opciones, confirmar o rechazar. En caso que se haya confirmado, se ejecutará el servicio de chat.

- *ChatService.java,* El servicio encargado del chat. Iniciará chats entre usuarios.

3.3.2. Implementación y funcionamiento de los servicios

En este apartado explicamos cómo hemos implementado los servicios en nuestra aplicación, y cómo funcionan internamente e interactúan con el servidor.
Hemos creado dos clases, una para cada servicio, ambas heredan de la clase Service y cada de ellas tendrá los atributos y métodos que necesite, además de los métodos en común para que funcionen correctamente. En este ejemplo, mostramos como hemos implementado el servicio de matching.

1. Al crearse la pantalla MainActivity, se inicializan los componentes , listeners y también se llama al método que inicializará los servicios.
   Básicamente, las partes que consideramos más interesante de la creación del servicio son las siguientes:

   Esta parte en especial, normalmente ejecutaríamos la tarea llamando al método execute() de la clase, pero nos dio problemas, nunca se llegaba a ejecutar porque entraba en conflicto con el thread de la interfaz de usuario, finalmente encontramos una manera de ejecutar la tarea, cambiando la llamada por la que mostramos.

```
//myTask.execute();
myTask.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, "");
```

   Figura 22. Fragmento de código de la llamada al método que ejecuta la tarea asíncrona del servicio.

Y la parte importante del servicio, que es donde se comprueba si hay coincidencia de objetos.
Primeramente llama al webservice y comprueba si hay objetos que coincidan, posteriormente monta un objeto JSON con los datos recibidos por el servidor, si esos datos son correctos, se creara un nuevo intent, con "DATA", como parámetro, y mediante el método putExtra de este intent, se añade la información del objeto, posteriormente se llama al método sendBroadcast con el intent recién creado.

```
try
{
    //Llamamos al webservice que nos indicará si hay objetos que coinciden
    String strReturn = sql.ExecuteQuery(URL_CHECK_FOUND_ITEMS, content);

    //Montamos un JSON con el retorno
    JSONObject jsonObject = new JSONObject(strReturn);

    //Cogemos cada atributo del objeto JSON
    String IDItemLost = jsonObject.getString("IDItemLost");
    String IDItemFound = jsonObject.getString("IDItemFound");
    String strType = jsonObject.getString("Type");
    String strColor = jsonObject.getString("Color");
    String strMaterial = jsonObject.getString("Material");
    String strDescription = jsonObject.getString("Description");
    String RowID = jsonObject.getString("RowID");


    //Si no ha habido ningún problema
    if((Integer.parseInt(IDItemLost) != 0) && (Integer.parseInt(IDItemFound) != 0))
    {

        //Cogemos todos los datos del objeto
        Intent intent = new Intent("DATA");
        intent.putExtra("IDItemLost", IDItemLost);
        intent.putExtra("IDItemFound", IDItemFound);
        intent.putExtra("Type", strType);
        intent.putExtra("Color", strColor);
        intent.putExtra("Material", strMaterial);
        intent.putExtra("Description", strDescription);
        intent.putExtra("RowID", RowID);

        //Enviamos un sendBroadcast para que el receiver coja los datos
        sendBroadcast(intent);

    }
```

Figura 23. Fragmento de código relevante dentro del servicio de matching.

2. Una vez inicializado el servicio, se inicializa el IntentFilter y el recibidor, además de crear un nuevo intent en el que se ejecutará el servicio.
   Lo comentamos en el siguiente fragmento de código.

```
//Inicializamos el servicio de matching
matchingService = new MatchingService(user);

//Inicializamos un intentfilter con la palabra "DATA"
matchingFilter = new IntentFilter("DATA");

//Inicializamos el receiver, será la clase encargada de
//realizar lo necesario cuando el servicio le avise, más adelante
//lo observaremos
matchingReceiver = new MatchingReceiver();

//Asignamos el intentfilter con el matchingreceiver
registerReceiver(matchingReceiver, matchingFilter);

//Cogemos el usuario actual y creamos un nuevo intent de la pantalla principal de
//la aplicación al servicio de matching, y ponemos el usuario como extras
user = matchingService.getUser();
Intent intentService = new Intent(MainActivity.this, matchingService.getClass());
intentService.putExtra("User", (Serializable) user);

//Arrancamos el servicio
startService(intentService);
```

Figura 24. Fragmento de código del método que inicializa las clases pertinentes para el correcto funcionamiento del servicio de matching.

3. Antes de lanzar el servicio, se han inicializado las clases que intervienen en la recepción de posibles datos que el servicio pudiera enviar.
   Definición de la clase MatchingReceiver (Figura X) con el método que se ejecutará cuando el servicio ejecute el método sendBroadcast(), lanzando la notificación de matching. Dicha notificación, básicamente se le añade un título y una acción al ser pulsada, que será, mediante un nuevo intent, ejecutar y mostrar la pantalla de confirmación de coincidencia objeto, *ItemMatchingActivity.java*, explicada en el apartado 3.3.1.5.

```
private class MatchingReceiver extends BroadcastReceiver {

    //Metodo que se ejecutara cuando el servicio llame al método
    //sendBroadcast(intent)
    @Override
    public void onReceive(Context arg0, Intent intent) {

        //Si no se ha leído la notificación
        //se lanzará
        if(!notificationsRead)
        {
            //Lanzamos la notificación de matching
            launchMatchingNotification(intent);

        }

    }
}
```

Figura 25. Fragmento de código del método que inicializa las clases pertinentes para el correcto funcionamiento del servicio de matching.

4. Una vez inicializados todos los elementos que participaran en la recepción, lanzamos el servicio mediante el método startService() y el servicio se ejecutará cada 60 segundos, consultando si hay objetos perdidos que coinciden con alguno que el usuario ha encontrado.

3.3.3. Activities

Las activities, es decir, todas las pantallas del proyecto, y algunos widgets que han sido utilizados para ayudar a visualizar los datos de manera correcta, se encuentran en el proyecto, en la carpeta llamada layout.

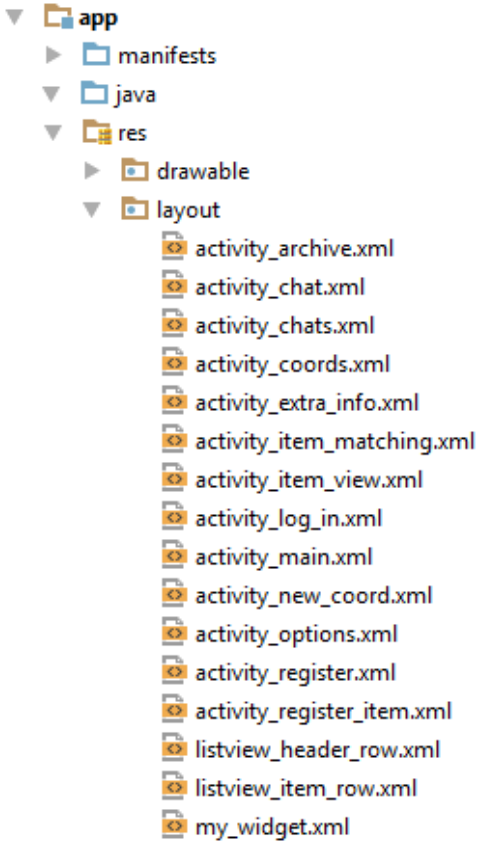

Figura 26. Estructura de todas las pantallas de la aplicación, situadas en el paquete layout.

3.3.3.1. Diagrama de activities

Una vez listadas las pantallas de la aplicación, hemos considerado que resultaría interesante mostrar en un diagrama resumido, cómo se puede acceder a las pantallas.

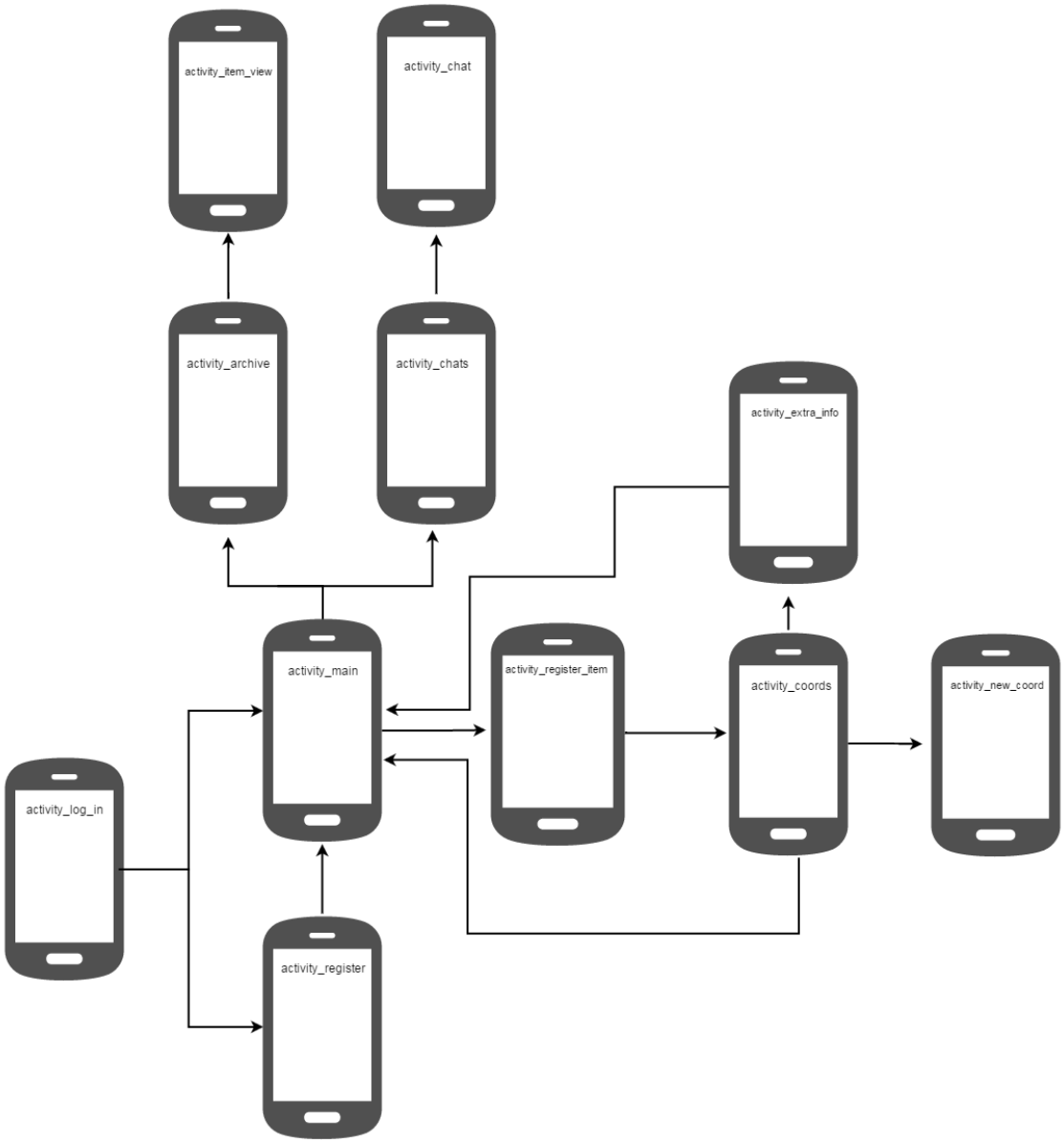

Figura 27. Diagrama de activities de la aplicación.

Para evitar congregación de flechas. hemos considerado la opción de evitar las flechas de retorno, indicando únicamente los sentidos interesantes y obviando la opción de volver atrás.

3.4. Desarrollo del servidor

La implementación del servidor la hemos dividido en varias carpetas. Cada carpeta contiene ficheros php para las diversas funcionalidades. Se ha dividido de esta manera, para tener una cierta concordancia con la división de paquetes de la parte del cliente, haciendo más fácil la legibilidad del código.

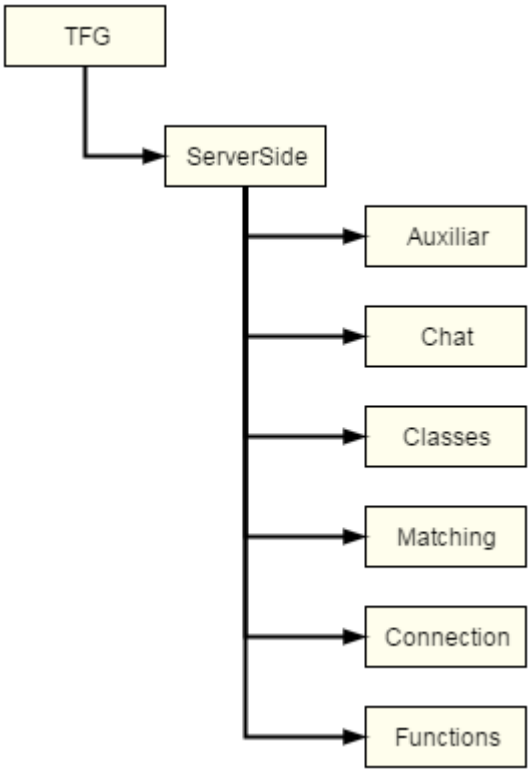

Figura 28. Estructura de las carpetas dentro del servidor.

3.4.1. Paquete Auxiliar

El paquete auxiliar, únicamente consta de una clase, aún así hemos decidido crear el paquete teniendo en cuenta posibles futuras ampliaciones.

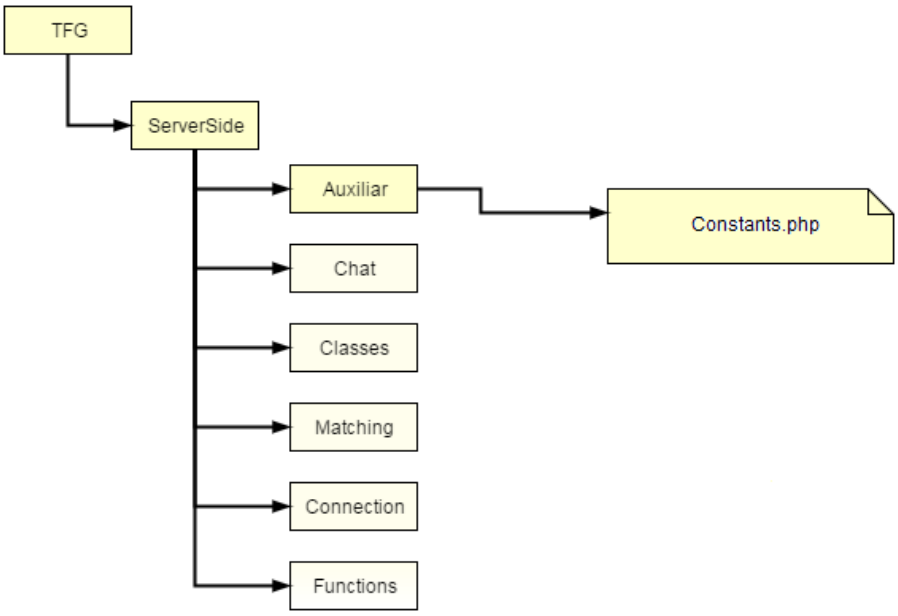

Figura 29. Estructura del paquete Auxiliar.

- *Auxiliar.php* que contiene constantes usadas en todo el código del servidor.

3.4.2. Paquete Chats

Contiene todas las clases que participan tanto en la creación de chats y comprobación de usuarios que participan en éstos.

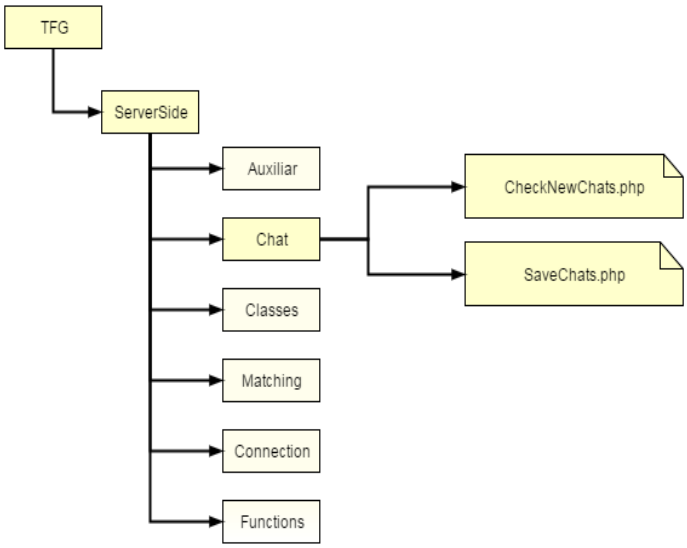

Figura 30. Estructura del paquete Chats.

- *CheckNewChats.php.* Webservice que llama el servicio de chats del cliente, comprobará mediante el email del usuario, le devolverá una lista de los chats que tiene con otros usuarios registrados en la aplicación.

- *SaveChats.php.* Funcionalidad que crea un chat y lo guarda en la base de datos, necesita el identificador de tabla tUsers de los dos usuarios que se chatearán.

3.4.3. Paquete Classes

Contiene, como hemos diseñado en la parte del cliente, las clases importantes de la aplicación, la clase del usuario y de los objetos con sus coordenadas.

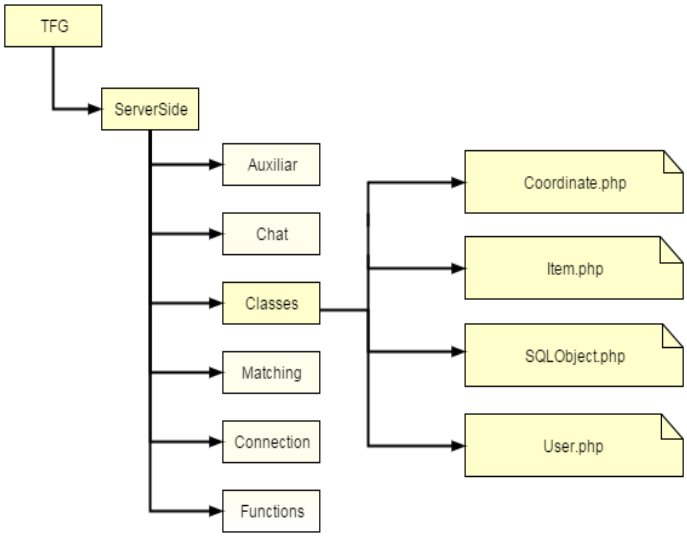

Figura 31. Estructura del paquete Classes.

- *Coordinate.php.* Clase contiene la información de una coordenada, la Xy la Y, además del método que la guarda en base de datos.

- *Item.php.* Clase que contiene la información de un objeto, tipo, marca, material, color, si se perdió o encontró y cuando, el estado, la descripción, la ID de la tabla de usuarios y sus coordenadas. También contiene el método que lo guardará en la base de datos además.

- *User.php.* Clase que contiene el correo, el nombre de usuario y la contraseña de un usuario junto a los métodos que guardan y devuelven la información del usuario.

- *SQLObjetct.php.* Clase padre para las clases anteriores, que contiene métodos genéricos y auxiliares.

3.4.4. Paquete Matching

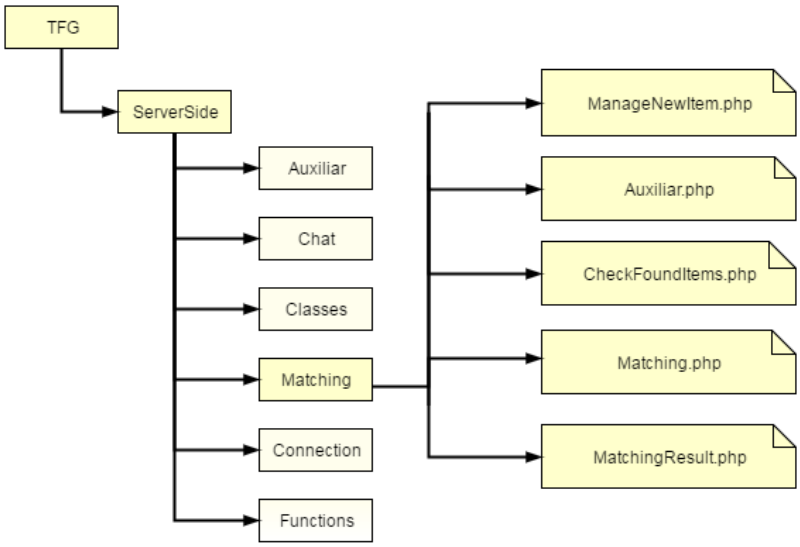

Figura 32. Estructura del paquete Matching.

- *Auxiliar.php,* Contiene métodos auxiliares para dar soporte al algoritmo de matching, ayudando a construir las consultas añadiendo condiciones.

- *CheckFoundItems.php,* es la clase que mirará si hay objetos en la tabla de items a comparar, y en caso que así sea, indicará al usuario que encontró el objeto, las características que introdujo el usuario que lo perdió, para que así éste decida si es o no es el mismo objeto.

- *MatchingResult.php,* recibirá vía Post el resultado de la decisión del usuario que encontró el objeto, en caso que el resultado sea positivo, se actualiza la tabla de comparar items, indicando el flag que ya la comparación finalizó y que fue positiva y se devuelve la información del usuario que perdió el objeto al usuario que lo encontró para que puedan comunicarse.

- En caso que el resultado fuera negativo, se eliminará la fila de la tabla a comparar, y se añadirá a la tabla de items que no coinciden, para que no se incluyan en búsquedas futuras.

- *Matching.php* Es la principal clase del algoritmo de matching, el funcionamiento del algoritmo consiste en:

    1. Se obtienen todos los objetos perdidos de la base de datos.
    2. Para cada uno de estos objeto, comparar con todos los objetos encontrados de la base de datos que cumplan:

1. Características físicas similares, tipo de objeto, color, material y marca (Figura 33).
2. Que no exista ya en la tabla de items a comparar ni tampoco en la tabla de items a no comparar (Figura 34).
3. Que alguna de las coordenadas de ambos objetos coincidan según una tolerancia (Figura 35).

Si todas estas condiciones se cumplen, se insertará en la tabla de items a comparar, el objeto perdido junto con el objeto encontrado, indicando que puede haber coincidencia en esos dos objetos.
En caso negativo, se insertará en la tabla de items a no comparar, es decir, no coinciden por algún motivo, y no tiene sentido volver a compararlos en un futuro.

```
//Construímos la parte de la consulta encargada del tipo de objeto
$strSQLType = $aux->construct_type_SQL($itemLost->ItemType);

//La parte que de la marca
$strSQLBrand = $aux->construct_brand_SQL($itemLost->Brand);

//La parte del material
$strSQLMaterial = $aux->construct_material_SQL($itemLost->Material);

//La parte del color
$strSQLColor = $aux->construct_color_SQL($itemLost->Color);

//La parte que gestiona cuando se perdió
$strSQLDate = $aux->construct_date_SQL($itemLost->When, $itemLost->RegisterDate);

//Y finalmente que cumpla las condiciones siguientes,
//un item que se haya encontrado y por otro usuario diferente
$strSQLFinalPart = " AND I.itemStatus = 0 AND I.foundlost = 'Found' AND I.UserID <> $itemLost->UserID ";
```

Figura 33. Fragmento de código donde se construye la consulta de las características físicas del objeto.

```
//Concatenamos los trozos de query construídos
$strSQL2 = $strSQLType . $strSQLBrand . $strSQLMaterial . $strSQLColor . $strSQLDate . $strSQLFinalPart;

//Y añadimos que el objeto no esté en la tabla tMatchingItems, es decir,
//que no haya una comparación con otro objeto existente y que no exista ya como un objeto
//que se comparó en el pasado y no hubo coincidencia, es decir, que no exista en tNonMatchingItems
$strSQL3 = " AND (SELECT COUNT(*) FROM tNonMatchingItems WHERE IDItemFound = $itemLost->ID) = 0 ";
$strSQL4 = " AND (SELECT COUNT(*) FROM tMatchingItems WHERE IDItemFound = I.ID) = 0 ";
```

Figura 34. Fragmento de código que restringe a objetos que no existan en las tablas tMatchingItems ni tNonMatchingItems.

```php
//Si las coordenadas coinciden, tenemos una coincidencia
if ($aux->coordinates_match($itemLost->CoordinatesList, $CoordinatesListFound) == 1)
{
    //Insertamos el par de IDs de item perdido y encontrado en la tabla de coincidencias
    //Marcando el flag de esperando confirmación a 1
    $strQuery = "INSERT INTO tMatchingItems (IDItemLost, IDItemFound, Waiting) VALUES ($itemLost->ID, $IDItem, 1)";
    $connection->ExecuteQuery($strQuery);
    $found = true
}
else
{
    //Si por el contrario, entramos en el else, no ha habido coincidencia
    //Introducimos el par de IDs perdido-encontrado en la tabla de no coincidencia
    $strQuery = "INSERT INTO tNonMatchingItems (IDItemLost, IDItemFound) VALUES ($itemLost->ID, $IDItem)";
    $connection->ExecuteQuery($strQuery);
}
```

Figura 35. Fragmento de código que comprueba si las ubicaciones son similares.

3.4.5. Paquete Connection

Contiene una única clase, aún así hemos considerado en crear un paquete, para diferenciar la función de la clase y teniendo en cuenta posibles ampliaciones en un futuro.

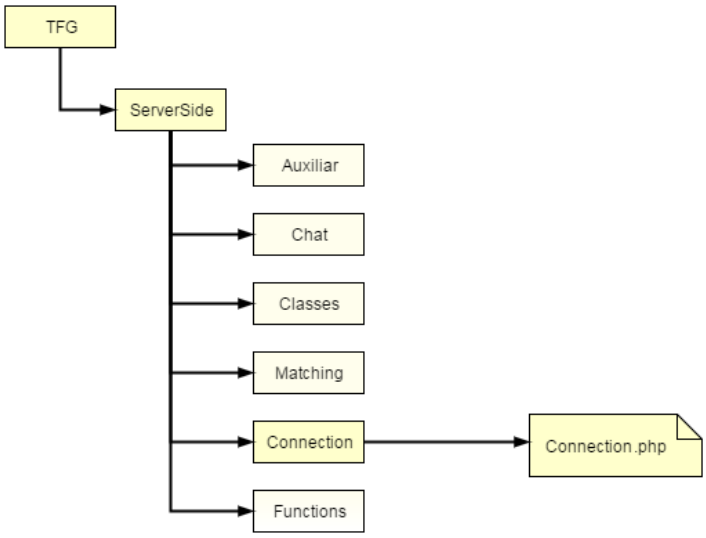

Figura 36. Estructura del paquete Connection.

- C*onnection.php* contiene todo lo necesario para que el servidor establezca la conexión con la base de datos. Necesita la URL del servidor, usuario y contraseña, si fuera necesario, y el nombre de la base de datos para establecer la conexión.

3.4.6. Paquete Functions

Contiene las principales funcionalidades de la aplicación, guardar y recuperar la información de un usuario y guardar un objeto.

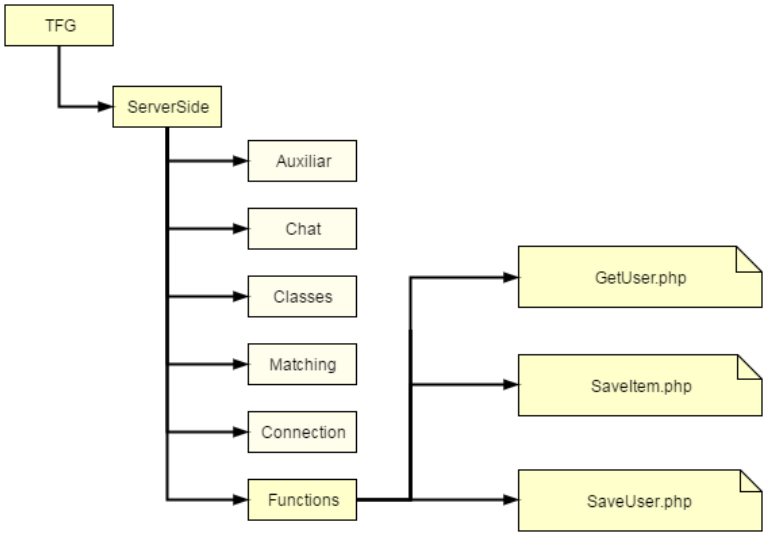

Figura 37. Estructura del paquete Functions.

- *SaveUser.php,* recibe vía post toda la información necesaria para guardar un usuario en la base de datos, correo, nombre de usuario y contraseña. Antes de guardar la contraseña, la encripta utilizando como función hash para encriptar sha512.

- *GetUser.php*, recibe vía post, el correo y la contraseña de un usuario, y si coinciden con un usuario registrado, obtiene toda los datos del usuario que hay guardados en la base de datos.
  Para comprobar si la contraseña coincide con la que hay encriptada y guardada en base de datos, se encripta utilizando sha512 y se compara el string resultante con el almacenado en base de datos, si ambos son iguales, es el correcto.

- *SaveItem.php,* recibe también vía post toda la información para guardar un objeto, recibe, el correo del usuario, para asociarlo a éste, tipo de objeto, color, material, marca, si perdió o encontró y cuándo, descripción del objeto, en caso de que fuera perdido, y el número y lista de coordenadas.

- Después de recoger todas las características del objeto, se crea un nuevo objeto de tipo Item y se llamará al método saveItem.

3.5. Diseño de la base de datos

Para el correcto funcionamiento de la aplicación es necesario un correcto diseño de la base de datos, ya que es el lugar donde se almacenará toda la información de la aplicación, le hemos dedicado especial atención.
Para ello hemos creado las tablas necesarias, en el siguiente apartado explicamos brevemente la función que desempeña cada una.

3.5.1. Modelo Entidad-Relación

Una vez entendido la utilidad de cada tabla, mostramos el modelo entidad-relación para entender cómo las tablas se relacionan entre si.
Resulta sencillo pensar que, un usuario puede registrar cualquier número de objetos, que a su vez, éstos contendrán de 1 a 3 coordenadas como máximo, y este usuario podrá chatear con cualquier otro número de usuarios.
Un objeto, a su vez, puede coincidir únicamente con otro objeto, mientras que puede no coincidir con cualquier número de objetos.

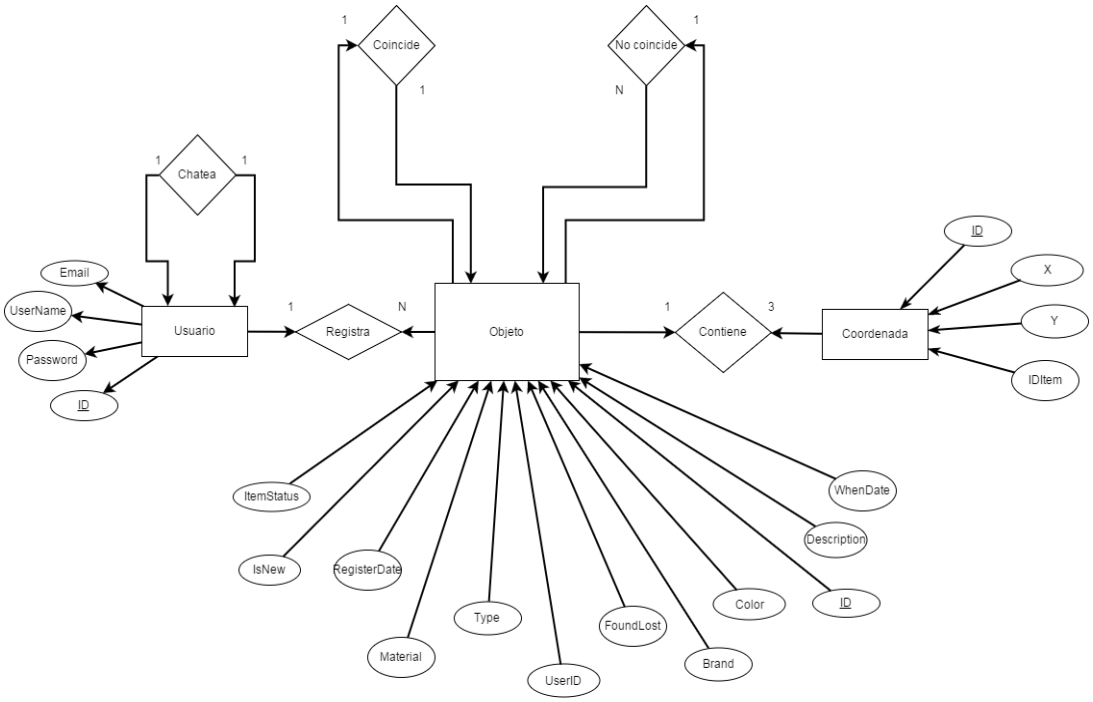

Figura 38. Modelo Entidad-Relación.

Resultando el siguiente diagrama con las tablas que finalmente necesitamos en la base de datos. Para las coincide o no coincide entre objetos, hemos diseñado dos tablas, podríamos haber diseñado una con un flag de coincidencia o no coincidencia, pero de este modo, separamos los conceptos diferentes en dos tablas.

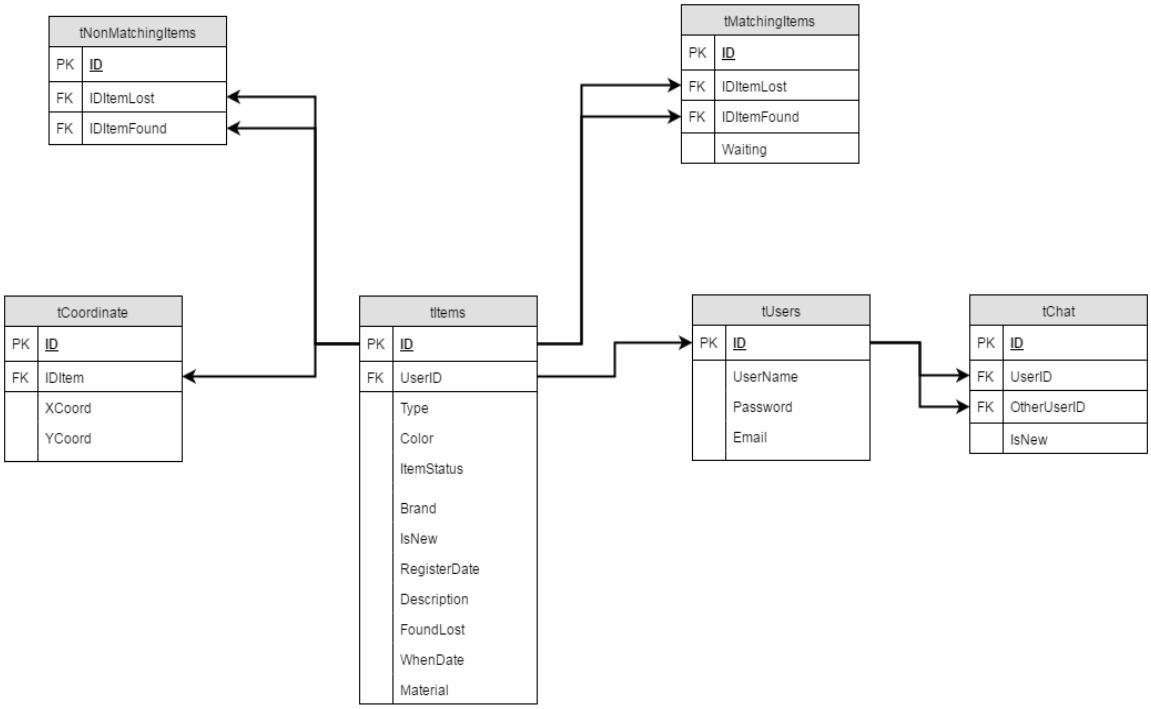

Figura 39. Modelo Entidad-Relación con las tablas finales.

3.5.2 Tablas de la base de datos

3.5.2.1. tUsers

Es la tabla donde se guarda toda la información de un usuario, es decir, el correo electrónico, el nombre del usuario y la contraseña, que se guarda encriptada en la base de datos.

3.5.2.2. tItems

La tabla que contiene toda la información de un objeto, contiene el tipo de objeto, el color, la marca, el material del cual está hecho, si es un objeto perdido o un objeto encontrado, cuando se perdió, la fecha de registro, el estado del objeto, la descripción proporcionada por el usuario que registró su pérdida, y la ID del usuario que hace referencia a la tabla tUsers.

3.5.2.3. tCoordinate

Tabla que contiene la información de una ubicación, que son las coordenadas X e Y, además de la ID del objeto guardado en la tabla tItems.

3.5.2.4. tMatchingItems

Es la tabla donde se guardan los IDs de aquellos objetos que el algoritmo determinó que fueron similares, además hay un flag que indica si el matching ha sido confirmado o no, por defecto es 1, cuando el usuario realice una acción, ya sea confirmar o rechazar que el objeto que posee coincide con el objeto perdido, este flag se pondrá a 0.

3.5.2.5. tNonMatchingItems

La tabla donde se guardan los objetos que no coinciden, para que no se tengan en cuenta en futuras comparaciones. Se puede guardar IDs de objetos de dos maneras, la primera cuando el algoritmo determina que no son similares y la segunda cuando el usuario rechaza la notificación de la aplicación. Contiene las IDs de los objetos que no coinciden.

3.5.2.6 tChat

La tabla donde se guardan los chats que mantienen los usuarios. Únicamente guarda los chats entre usuarios, no guarda la conversa. Contiene las IDs de los usuarios los cuales tienen un chat activo entre ellos.

4. Discusión

Respecto a las funcionalidades de la aplicación, por limitación de tiempo y de conocimientos técnicos en algunos aspectos concretos, la funcionalidad que permite a dos usuarios ponerse en contacto una vez ha habido coincidencia entre la pareja de objeto perdido y objeto encontrado que registraron no se terminó completamente. En vez de pararnos en este aspecto, decidimos centrarnos en otras funcionalidades que también eran muy relevantes para la aplicación.
De hecho, el resto de funcionalidades que se planearon sí que se han implementado satisfactoriamente.
A continuación discutimos una serie de aspectos que fueron surgiendo a lo largo de todo este tiempo y cómo se han tratado.

4.1. Aspectos de desarrollo

Durante el desarrollo de la aplicación, nos hemos encontrado con una serie de desafíos técnicos, algunos los hemos solucionado y para otros, la solución que decidimos tomar, aunque seguramente no sea la más adecuada, nos permitió desbloquear posteriores implementaciones o funcionalidades principales.
Por otro lado hay cuestiones técnicas que, aunque las hemos identificado, como problemas, no las hemos podido solucionar en este TFG. Se dejan como perspectivas de futuro, que resumimos más adelante.

4.1.1. Desafíos técnicos solucionados

En este apartado mostramos los principales desafíos técnicos que surgieron durante la implementación y que más tiempo llevó afrontarlos y solucionarlos.

1. La introducción de la localización al añadir una nueva coordenada a un objeto era una de las cuestiones importantes, debía ser sencillo e intuitivo. Lo implementamos de tal manera que el puntero está fijo en el centro del mapa y el usuario únicamente debe mover el mapa y cuando esté satisfecho con la ubicación solamente deberá aceptar. El puntero no siempre se sitúa exactamente en el centro del mapa, pero lo consideramos un aspecto menor, ya que la localización puede realizarse correctamente.
2. Las notificaciones dieron problemas también, ya que quisimos trasladar el método al paquete auxiliar, pero para lanzarla se requiere que dicho método esté implementado en una activity, por lo que tuvimos que definir el método en la pantalla principal para poder ser correctamente ejecutado.
3. Se requería de servicios para el correcto funcionamiento de la aplicación, inicialmente se planteó un paquete para éstos, pero por problemas en el fichero

manifest.xml de en las rutas de los paquetes los servicios se movieron finalmente en el paquete principal.

4. Las ListViews permiten crear una lista de elementos en una pantalla, pero únicamente permiten mostrar texto, de manera que buscamos una alternativa a esto. Encontramos una forma, creando un widget, de mostrar además del texto un icono para mostrar el estado del objeto, y si era un objeto perdido o encontrado.
5. Otro problema fue cómo generar la consulta teniendo en cuenta los campos informados de un objeto registrado. Si un usuario que encuentra un objeto, no sabe, la marca del objeto, ésta no debe ser una condición de búsqueda, ya que si así fuera, solamente buscaría objetos sin marca.

   Pensamos que la mejor manera sería ir generando la consulta dinámicamente en base a los campos informados de un objeto, teniendo en cuenta además la diversidad de elementos existentes dentro de un mismo campo, es decir, alguien que registra la pérdida de un objeto metálico, puede ser registrado por quien lo perdió como si fuera un objeto de aluminio o de acero, por lo tanto tuvimos que añadir sinónimos o palabras de muy similar significado para evitar esta controversia.
6. A la hora de registrarse en una aplicación, si la aplicación es de carácter global, se deben tener en cuenta los caracteres especiales u otros diccionarios. Es muy probable que dichos caracteres, dependiendo de la codificación de la base de datos, generen algún problema más adelante, por lo que hemos añadido, usando expresiones regulares, una restricción a su uso, permitiendo los siguientes caracteres en los campos de registro de usuario, a-A, z-Z y números del 0 al 9.

   Además, al informar el correo electrónico, se debe seguir la estructura tal que, correo@electronico.x, siendo x el dominio del correo de como máximo 3 caracteres.
7. El problema legal, uno de los principales problemas de la aplicación. El Código Civil español regula el modo de actuar frente a un hallazgo, y, además, existe una regulación en el Código Civil de Catalunya que modifica el español. Expusimos nuestra idea de aplicación a entendidos sobre la materia y nos comentaron que la idea de la aplicación era buena y no rompía ninguna regulación del código civil, pero nos aconsejaron que, de algún modo, hiciéramos referencia al apartado sobre los hallazgos del código civil.

4.1.2. Problemas técnicos pendientes de solucionar

1. Uno de los principales problemas que nos hemos durante la implementación ha sido el modo de poder desarrollar callbacks enviados desde el servidor al cliente para poder notificarlo de eventos que sucedan. Ha sido imposible encontrar un modo de implementarlos, por lo que aquellas partes que era necesario de un callback se ha tenido que pensar de otra manera, en general, permitiendo la comunicación asíncrona, y ya que ésta no ha podido ser, hemos tenido que crear servicios que, mediante polling, consultaran a servidor si había algún cambio a tener en cuenta. Esto claramente no es la manera óptima de programar ya que a nivel de la aplicación afecta en el gasto de los recursos.
2. Una importante herramienta de la aplicación es el chat. Desgraciadamente no ha sido posible implementarlo con éxito debido al problema mencionado en el apartado anterior. El chat entre dos usuarios es un buen ejemplo de comunicación asíncrona, una posible solución hubiera sido utilizar como referencia el uso de servicios para hacer que funcionara el chat, pasando por servidor el mensaje y enviándolo al usuario pertinente, pero antes de hacer una implementación un poco engorrosa, decidimos dejar el chat apartado y centrarnos en otros puntos también importantes.
3. En el apartado 4 de los problemas solucionados hemos comentado como poder crear elementos en los ViewLists que puedan mostrar más elementos que solamente texto. Todo ha funcionado correctamente menos la visualización de los iconos.
4. El apartado 5 del punto 4.1 Solucionados, habla de cómo montar la consulta de manera dinámica. Inicialmente se nos ocurrió el uso de stored procedures para agilizar y liberar de carga el servidor, ya que son métodos guardados en base de datos ejecutados por el motor de ésta. Pero por el motivo de la limitación del tiempo decidimos crear las consultas de manera dinámica como ya hemos explicado.

5. Conclusiones

Finalmente en el apartado de conclusiones hacemos una reflexión del proyecto, de todos los aspectos que nos hemos ido.

En primer lugar nos gustaría comentar el aspecto legal, relacionado con el código civil, el cual no tuvimos en cuenta inicialmente y, de cómo, una sencilla aplicación que pretende facilitar la recuperación de un objeto perdido, puede incluso proponer problemas con la legalidad vigente. Un aspecto, a priori absurdo, pero que podía suponer un drástico cambio de rumbo de nuestra idea, e incluso llegar a suponer el pensamiento de dejar a un lado el proyecto.

En cuanto a la etapa de búsqueda de requerimientos de la aplicación, gracias a las entrevistas, pudimos confirmar que nuestra app puede realmente ayudar a las personas a gestionar un aspecto importante en sus vidas cotidianas.

Antes de ponernos a implementar, decidimos organizarnos y, mediante el uso de Scrum, empezamos a planificar las tareas, pero pronto nos dimos cuenta que, Scrum es bueno para grupos de trabajo, pero no demasiado para un trabajo de desarrollo bastante individual. Por tanto, tomamos la decisión de planificar las tareas por funcionalidades, dedicarnos la totalidad del tiempo y enfocar el esfuerzo a una sola tarea hasta finalizarla y dedicarnos a la siguiente, de este modo nos asegurábamos que las tareas completadas, muy probablemente contendrían bajo número errores y podríamos incluso reutilizar código con seguridad.

Una vez acabada y también durante la implementación, fueron surgiendo ideas y nuevas posibilidades para añadir a la aplicación, que pueden ser implementadas en futuras versiones. Más adelante las explicamos de forma más detallada.

5.1. Mejoras a realizar en siguientes versiones

1. Uso de stored procedures en el servidor, métodos guardados en y, ejecutados por, la base de datos, de esta manera se libera carga del servidor.
2. El uso de callbacks en la aplicación para ahorrar recursos al cliente y poder implementar funcionalidades.
3. Añadir la funcionalidad del chat una vez los callbacks estén correctamente implementados.
4. Uso de captchas para evitar el fraude, sería interesante que en el último paso de registro de un objeto se mostrara el captcha, y el usuario, que al aceptarlo, le

permita registrar el objeto con éxito.

5. Modificar la información mostrada en los elementos del ListView de la pantalla de archivo para que sea más útil.
6. Mejoras en la interfaz para que sea más user-friendly, así como la ayuda del autocompletado en los campos de las características físicas del objeto a registrar.
7. Permitir registro de usuarios mediante Facebook o Google+.
8. Añadir el campo de Ciudad de residencia y País de nacimiento, entre otros, en la pantalla de registro de usuarios, de este modo, al añadir coordenadas para un objeto, inicializar la posición inicial en dicha ciudad por defecto.
9. Añadir el botón de ubicación actual en la pantalla que añade una coordenada para facilitar al usuario la introducción de ésta.
10. Añadir la utilidad de adjuntar una fotografía del objeto perdido al registrarlo.
11. Al añadir coordenadas, guardar la última ubicación añadida, facilitando también la inserción de posteriores coordenadas.
12. Modificar en cliente, servidor y base de datos la clase coordenada, añadiendo dirección completa en base a latitud y longitud, siendo así más intuitivo para el usuario que las coordenadas de la ubicación.
13. Permitir añadir objetos de más de un color y de más de un material a un usuario al registrar un objeto.
14. Permitir que, un objeto registrado pueda modificarse o eliminarse.
15. Añadir feedback de los usuarios, es decir, si dos usuarios se pusieron en contacto porque la pareja objeto perdido - objeto encontrado coincidía, dichos usuarios deberían indicarlo al sistema de algun modo.

De manera más específica, también pensamos en mejoras para el algoritmo de búsqueda:

1. Modificar el algoritmo para que tengan más prioridad los objetos que tengan más características informadas sobre los que vienen menos informados.
2. La tolerancia que se utiliza para determinar si la localización de dos objetos es similar debería aumentar en el tiempo, se debería de comprobar la fecha de registro con la actual y modificar la tolerancia en base a este parámetro, ampliando así el rango de búsqueda.
3. Pensamos también en cambiar el formato del algoritmo, es decir, actualmente es una consulta a una base de datos con unos filtros que dependen de las características de cada objeto, pero una posible mejora sería utilizar clasificadores. Transformar los objetos en vectores de características y en base a éstas, encontrar objetos similares.

6. Referencias

Página donde encontré como añadir iconos y más información en los elementos de un ListView.

- http://www.ezzylearning.com/tutorial/customizing-android-listview-items-with-custom-arrayadapter

Expresiones regulares para Java.

- http://www.mkyong.com/tutorials/java-regular-expression-tutorials/

Información genérica sobre la API de Android, dudas sobre funciones, modos de implementar según que funciones y información de PhP.

- https://developer.android.com/index.html
- http://es.stackoverflow.com/
- https://secure.php.net/manual/es/index.php

Información sobre la implementación de servicios.

- http://www.imaginaformacion.com/implementa-un-service-boot-en-tu-aplicacion-de-android/

Código civil de España.

- http://noticias.juridicas.com/base_datos/Privado/cc.html

Código civil de Cataluña.

- http://civil.udg.es/normacivil/cat/CCC/ES/L5-2006.2.htm#t4

Diseño de los mockups.

- http://ninjamock.com/

7. Anexo

Tabla de contenido

7.1. Resultado de las entrevistas

Pregunta	Usuario 1	Usuario 2	Usuario 3
¿Has perdido alguna vez algún objeto?	Si	Si	Si
¿Qué objeto?	Bufanda	Sudadera	Paraguas
¿Lo has encontrado?	No	No	Ha perdido muchos, los encontró todos menos uno.
¿Has hecho algo para encontrarlo?	No	Si, preguntar por donde creyó por donde lo perdió	No
¿Te has encontrado alguna vez algun objeto perdido por la calle ?	Si	Si	Si, Carteras, DNIs, etc..
¿Qué has hecho con ese objeto ?	Llevarlo a objetos perdidos	Se lo regaló a alguien	Guardarlo
¿Si pudieras perder 5 minutos de tu tiempo para facilitar que quien lo ha perdido lo pudiera recuperar, lo harías?	Si	Si, supongo que si	Si
¿Cómo lo harías?	Llevarlo a objetos perdidos	Lo llevarías a los mossos	Colgar un cartel cerca de donde se perdió
¿Conoces alguna aplicación para encontrar objetos perdidos ?	No	No	No
¿Si hubiera una aplicación para móvil que ayudara a la gente a encontrar sus objetos perdidos, la utilizarías?	Si	Si	Si
¿Qué te parecería una aplicación como esta?	Molt bé	Bien, guay	Útil
[Explicar mi aplicación]			
¿La usarías?	Si	Si	Si
¿Le añadirías o le quitarías algo?	Ara mateix no		No, está bien así

Pregunta	Usuario 4	Usuario 5
¿Has perdido alguna vez algún objeto?	Si	Si
¿Qué objeto?	Bolso con cosas	Llaves de casa
¿Lo has encontrado?	No	No
¿Has hecho algo para encontrarlo?	Fue a la policía a poner la denuncia	No
¿Te has encontrado alguna vez algun objeto perdido por la calle ?	Si	Si
¿Qué has hecho con ese objeto ?	Se lo quedó	No recuerda
¿Si pudieras perder 5 minutos de tu tiempo para facilitar que quien lo ha perdido lo pudiera recuperar, lo harías?	Sí, porque no	Si
¿Cómo lo harías?	Llevarlo a la policía	Llevarlo a la policía
¿Conoces alguna aplicación para encontrar objetos perdidos ?	No	No
¿Si hubiera una aplicación para móvil que ayudara a la gente a encontrar sus objetos perdidos, la utilizarías?	Si	Si
¿Qué te parecería una aplicación como esta?	interesante	Está bien
[Explicar mi aplicación]		
¿La usarías?	Si	Si
¿Le añadirías o le quitarías algo?	Está bien	Está bien

7.2. Diseño en papel de las principales pantallas

7.2.1. Pantalla principal de la aplicación

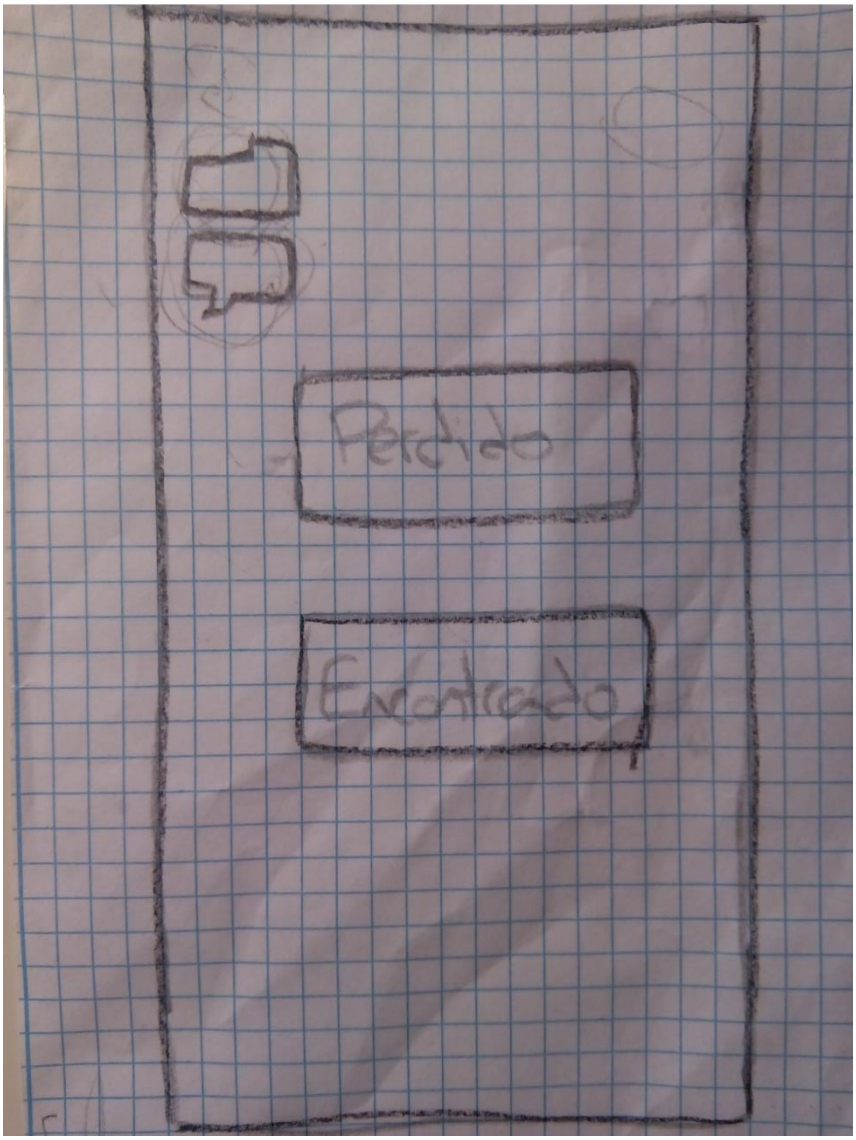

Figura 40. Pantalla principal de la aplicación

7.2.2. Pantalla de registro de objetos

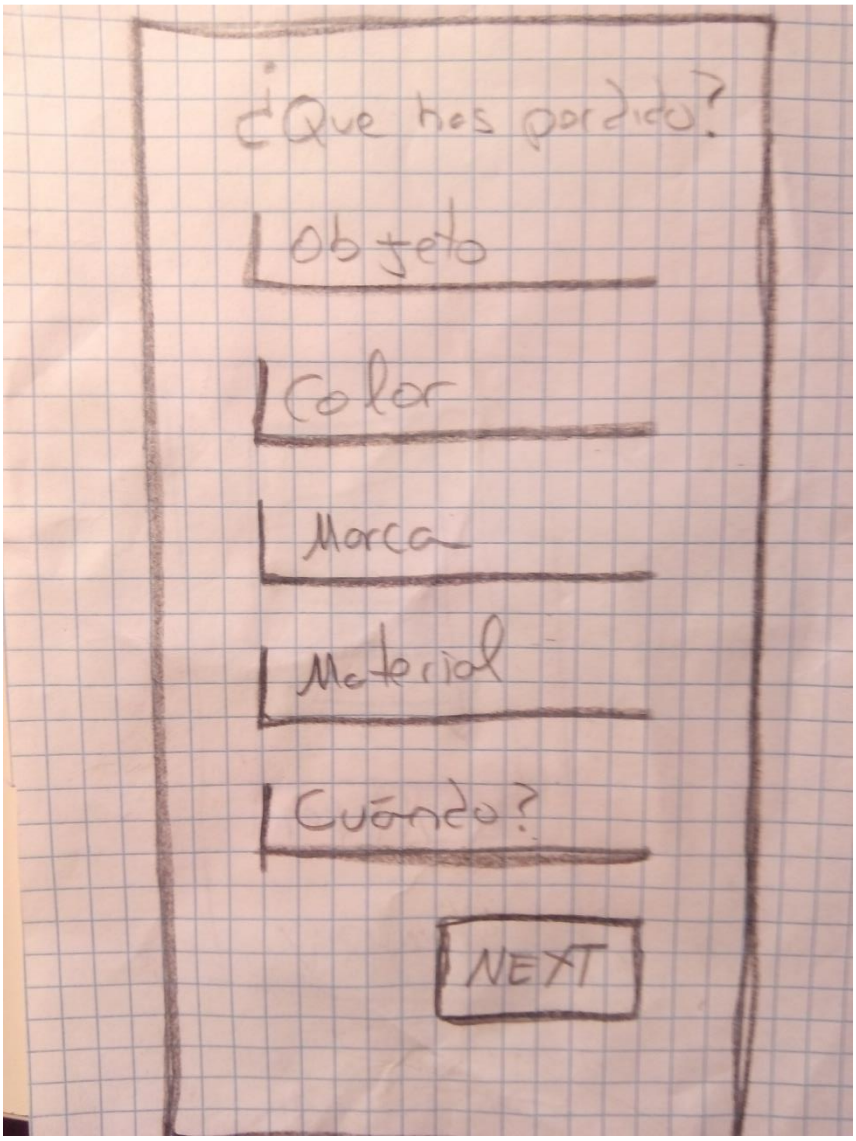

Figura 41. Pantalla de registro de objetos

7.2.3. Pantalla de gestión de las coordenadas.

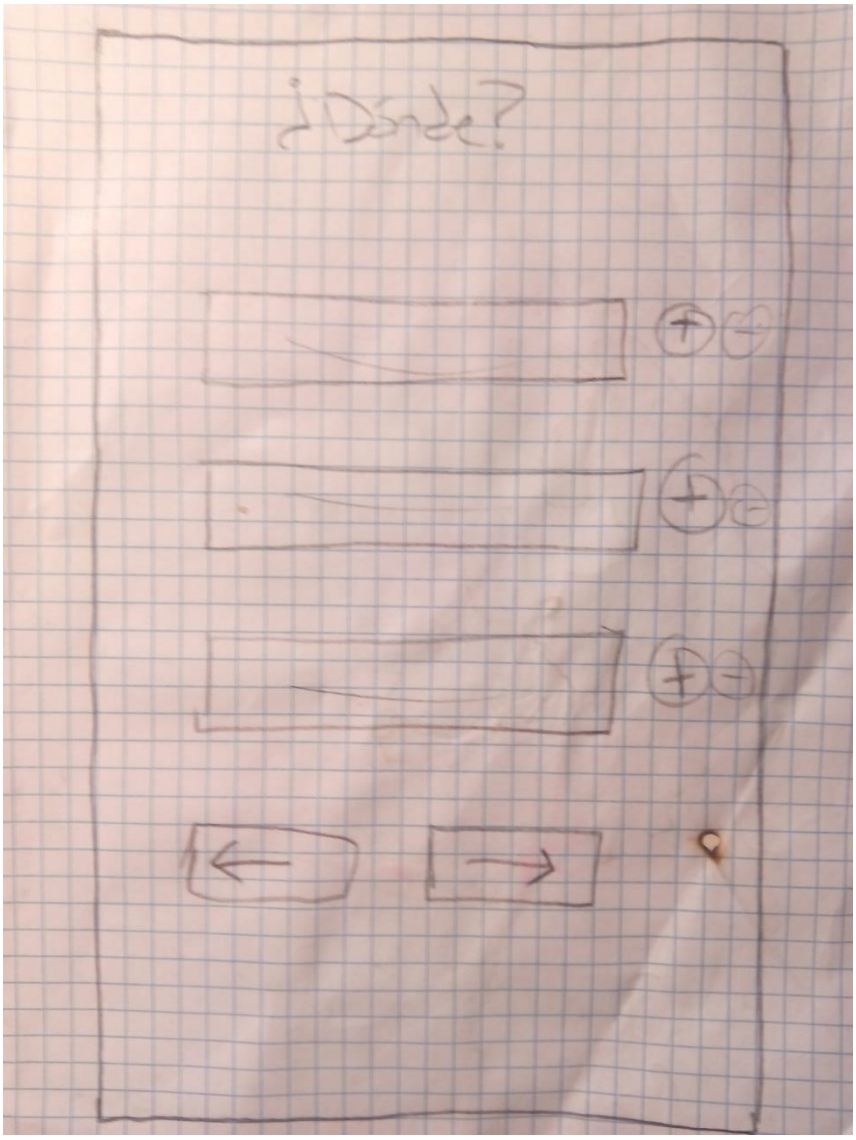

Figura 42. Pantalla de gestión de las coordenadas.

7.2.4. Pantalla de introducción una nueva ubicación.

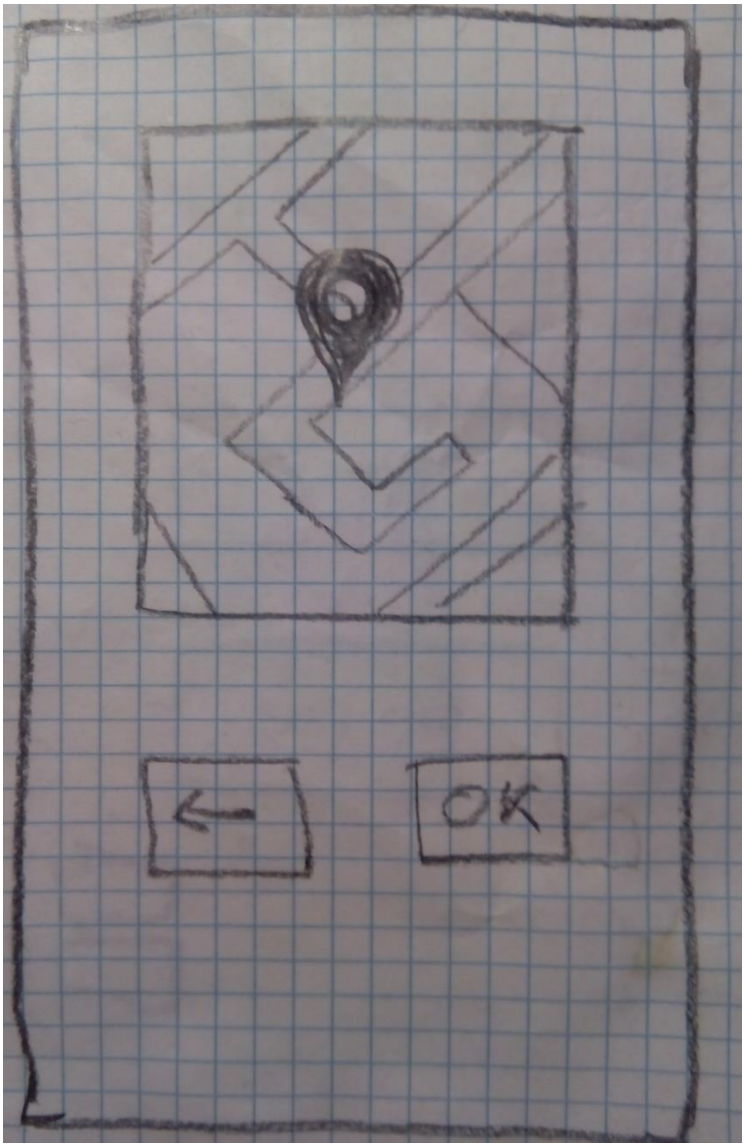

Figura 43. Pantalla de introducción de una nueva ubicación.

.

.2.5. Pantalla de introducción de información adicional

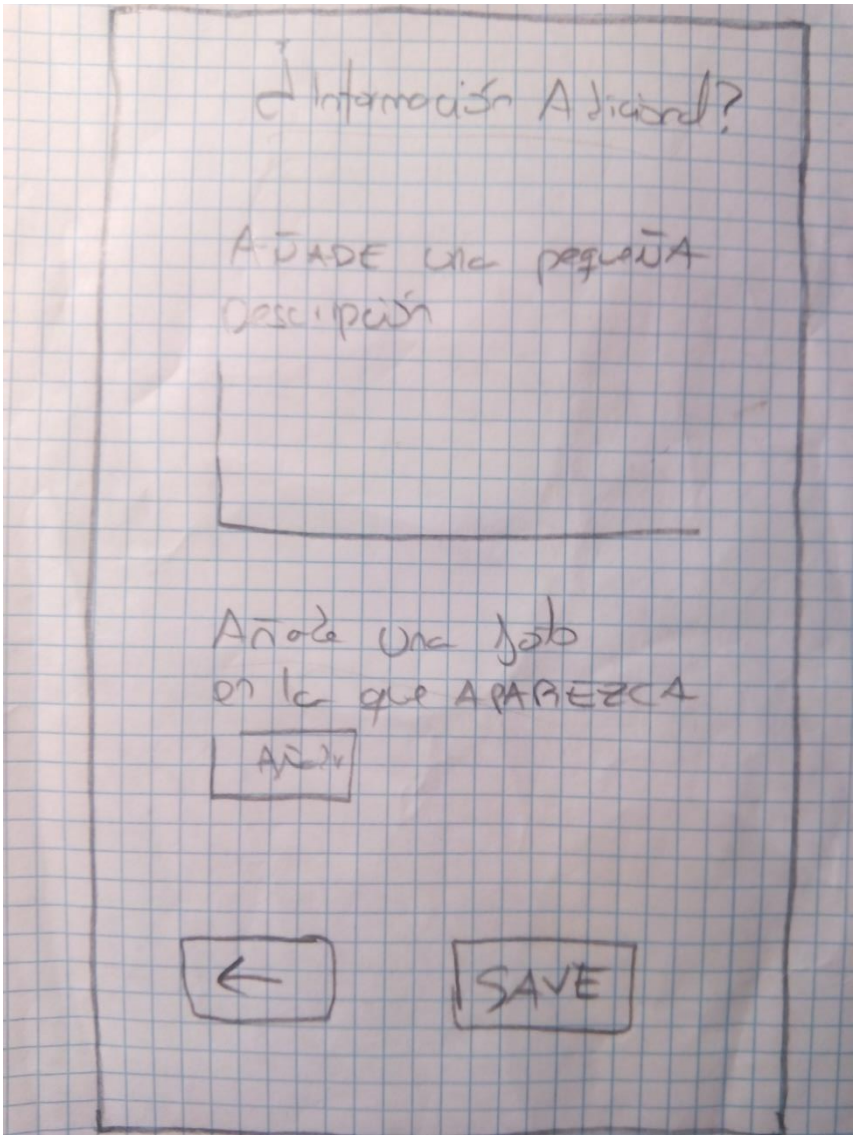

Figura 44. Pantalla de introducción información adicional de un objeto perdido